

ntor(s): SHAFFER ELIZABETH EVE; MA SHIHWIN; FAHRNI JIMOTH

icant(s): VENTRO CORP (US) ;

cation Number: WO2001US25468 20010815 ;

ity Number(s):

US20010274595P 20010310; US20010278558P 20010323; US2001028  
20010330; US20010917968 20010730 ;

Classification: G06F17/60 ;

valents: ;

TRACT:

present invention provides tools (414) and techniques for automatically generating configuration materials for an electronic commerce marketplace. An express specification (406) describes product categories and attributes in a form which is both readable by people and suitable for parsing (502) by software. The software extracts (504) information from the express specification and uses it to automatically generate configuration materials such as user interface configuration files (700), search interface configuration files (704), product catalog configuration files (712), quality assurance checklists (718), connection script frameworks (726), database load script frameworks, graphical data configuration file (728)s, and drafts of supplier documentation (730).

(19) World Intellectual Property Organization  
International Bureau



(43) International Publication Date  
19 September 2002 (19.09.2002)

PCT

(10) International Publication Number  
**WO 02/073493 A1**

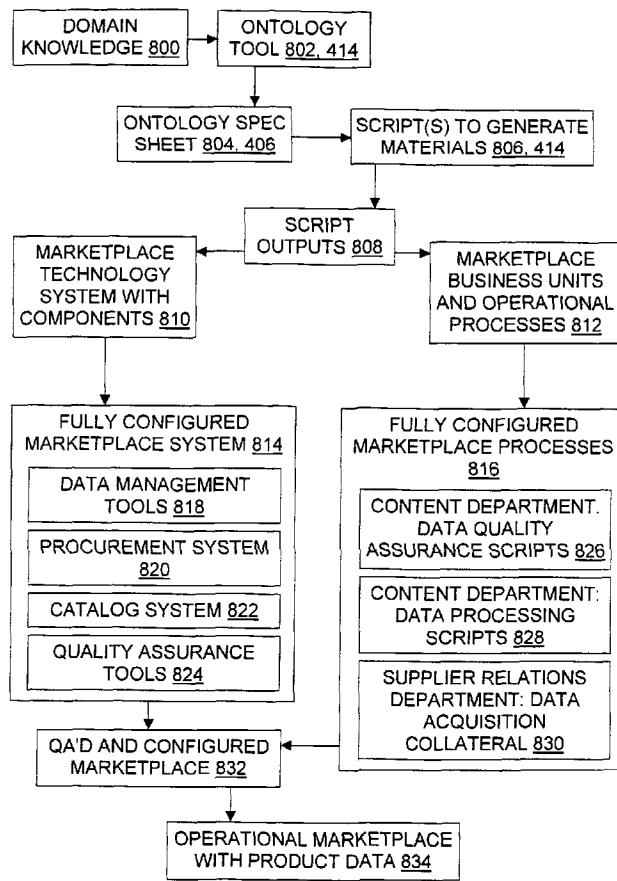
- (51) International Patent Classification<sup>7</sup>: **G06F 17/60** (72) Inventors: **SHAFFER, Elizabeth, Eve**; 330 North Mathilda Avenue, Apt. #108, Sunnyvale, CA 94085 (US). **MA, Shihwin**; 2038 Santa Cruz Avenue, Menlo Park, CA 94025 (US). **FAHRNI, Jimothy, Allan**; 1623 West Selby Ln., Redwood City, CA 94061 (US).
- (21) International Application Number: PCT/US01/25468
- (22) International Filing Date: 15 August 2001 (15.08.2001)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
- |            |                            |    |
|------------|----------------------------|----|
| 60/274,595 | 10 March 2001 (10.03.2001) | US |
| 60/278,558 | 23 March 2001 (23.03.2001) | US |
| 60/280,196 | 30 March 2001 (30.03.2001) | US |
| 09/917,968 | 30 July 2001 (30.07.2001)  | US |
- (74) Agent: **OGILVIE, John, W., L.**; Computer Law++, 1211 East Yale Avenue, Salt Lake City, UT 84105 (US).
- (81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(71) Applicant: VENTRO CORPORATION [US/US]; 1500 Plymouth Street, Mountain View, CA 94043 (US).

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

[Continued on next page]

(54) Title: FACILITATING ELECTRONIC COMMERCE MARKETPLACES BY AUTOMATICALLY GENERATING FILES FROM A STRUCTURAL ONTOLOGY SPECIFICATION



(57) Abstract: The present invention provides tools (414) and techniques for automatically generating (506) configuration materials for an electronic commerce marketplace. An express structural ontology specification (406) describes product categories and attributes in a form which is both easily read by people and suitable for parsing (502) by software. The software extracts (504) ontology information from the express specification and uses it to automatically generate configuration materials such as user interface configuration files (700), search interface configuration files (706), product catalog configuration files (712), quality assurance checklists (718), data extraction script frameworks (726), database load script frameworks, graphical data entry tool configuration file (728)s, and drafts of supplier documentation (730).



patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

**Published:**

— *with international search report*

**FACILITATING ELECTRONIC COMMERCE MARKETPLACES  
BY AUTOMATICALLY GENERATING FILES  
FROM A STRUCTURAL ONTOLOGY SPECIFICATION**

5

**RELATED APPLICATIONS**

The present application claims priority to, and incorporates by reference, the following United States provisional applications: serial no. 60/274,595 filed March 10, 2001, serial no. 60/278,558 filed March 23, 2001, and serial no. 60/280,196 filed March 30, 2001.

10

**COPYRIGHT NOTICE**

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

15 The copyright owner does not hereby waive any of its rights to have this patent document maintained in secrecy, including without limitation its rights pursuant to 37 C.F.R. § 1.14.

**FIELD OF THE INVENTION**

The present invention relates generally to the creation and maintenance of computer network sites, and relates more particularly to tools and techniques for automatically generating files that are used in configuring commercial web sites.

**TECHNICAL BACKGROUND OF THE INVENTION**

Various approaches for generating commercial web sites are known. For instance, 25 Figure 1 illustrates a prior approach like that discussed in the document titled "Welcome to eCommerce Tools!", which was provided at pages 20 through 48 of incorporated provisional application serial no. 60/274,595 filed March 10, 2001. The material described in said eCommerce Tools document is not, in and of itself, claimed as the present invention, but the eCommerce Tools document is part of the present application for purposes such as understanding the state of the art. The catalog 100 is assumed to be a conventional catalog such as a paper catalog; it may also be in electronic form, such as in word processor files. A structural ontology of the web site 108, tools 104 used to generate the site 108, manual data entry 102, and automatic file generation 106 are discussed in particular at the eCommerce Tools document's internal pages 18-28. Apparently the structural ontology of the web site 108 exists 30 as a whole only implicitly in the internal code and data structures of the eCommerce Tools 35

software. There is apparently no express structural ontology specification which is human-readable by non-programmers, and which is easily copied (and possibly modified) for use in configuring several (possibly somewhat different) web marketplaces at about the same time. The focus of the approach shown in Figure 1 is instead on a single web site for a single vendor.

5       Figure 2 illustrates an approach such as the approach that is discussed in the document titled "Chemdex™, a vortex business", which was provided at pages 49 through 51 of incorporated provisional application serial no. 60/274,595 filed March 10, 2001. The material described in said Chemdex™ document is not, in and of itself, claimed as the present invention, but the Chemdex™ document is part of the present application for purposes such as  
0 understanding the state of the art. In this approach several vendors' catalogs 200-204 are integrated into a shared structural ontology 206 through a process 212 that generally involves negotiations between the parties to reach agreement on the details of the shared structural ontology 206, as well as automatic and manual extraction of product data and entry thereof into a product database. Tools 208 are used to generate HTML pages, and graphical data entry tools  
5 208 may be used to automatically populate a product database. The structural ontology 206 is apparently implicit in the code and internally used data of such a tool 208, rather than being an express structural ontology specification which is human-readable by non-programmers and which is easily used in several (possibly somewhat different) online marketplaces at the same time. The focus of the approach shown in Figure 2 is thus on a single marketplace, such as one  
.0 for a particular industry; for instance, the Chemdex™ web site focused on commerce in products used for life sciences research.

Figure 3 illustrates an architecture involving transactional ontologies, to help clarify the important difference between a transactional ontology and a structural ontology. As indicated, a transactional ontology is concerned with the data formats and procedures used to perform  
5 transactions between parties in a marketplace. Early transactional ontologies often involved Electronic Data Interchange (EDI) format specifications. More recently, eXtended Markup Language (XML) Document Type Definitions (DTD) and other XML or XML-related format specifications are being used to implement transactions in accordance with an agreed-upon transactional ontology. By contrast, a structural ontology is concerned primarily with the  
0 products being offered, with their attributes, and with their relations to one another through grouping into categories, for instance. That is, transactional ontology focuses on commercial transaction procedures such as how products are ordered and paid for, while structural ontology focuses on web site structures such as those that specify which product information is presented to potential buyers, and how products relate to each other.

Ventro Corporation, assignee of the present invention, assisted with and ultimately owned assets of, Promedix.com, Inc. Promedix.com operated a web marketplace generally resembling the Chemdex™ web site discussed above. In particular, Ventro Corporation is assignee of United States application serial no. 09/496,361 filed February 1, 2000 for  
5 Promedix.com Corporation, an application which discusses transactions in a hub & spoke architecture and which is incorporated herein as background to provide additional detail regarding systems that generally resemble the system of Figure 3. Figure 3 shows “prior art” with respect to the invention of the present application; statements made here regarding Figure 3 and/or transactional systems and methods are meant to be understood in the context of this  
10 present application, not in any other application.

Prior to the earliest priority date claimed above, Ventro Corporation has provided services to help build various marketplaces, including for example web sites operated by Amphire Solutions, Broadlane, Chemdex, Industria Solutions, MarketMile, and Promedix.com (marks of their respective owners). Spreadsheet files which are similar or identical in form to  
15 express structural ontology specifications 406 discussed below were known to at least inventor Ms. Shaffer prior to the earliest priority date claimed above. Such spreadsheet files, which were also known as “templates”, were used by Ventro Corporation at least as early as August 25, 1999 and may have been disclosed outside Ventro at least as early as February 29, 2000. However, they were not then used as a basis for scripts to operate on to generate engineering  
20 configuration files as called for by the present invention. In particular, Figure 8 of incorporated provisional application no. 60/274,595 shows a printout of a spreadsheet which is in a format closely resembling that of an early version of the ontology specification 406. This spreadsheet was created on or about August 25, 1999. It was apparently only used internally by Ventro Corporation and was not used as input to scripts for automatically generating 506 engineering  
25 configuration files.

References which mention or discuss tools and techniques for facilitating electronic commerce are identified and discussed relative to the present invention in a Petition for Special Examining Procedure filed concurrently with the present application (or a U.S. counterpart). To the extent that the Petition describes the technical background of the invention, it is  
30 incorporated here by this reference.

Building a marketplace may be a complex project drawing on the different abilities of many people. Prior to the present invention, a variety of problems arose. Because configuration files were generated by hand and different people participated in different marketplace projects, it was sometimes difficult to determine who was responsible for providing particular

configuration files. Manually converting data (even from a generally agreed-upon specification) into engineering configuration files was also a tedious process, and thus subject to errors and inconsistencies. Version control was difficult, and version inconsistencies were sometimes difficult to detect. As a result, changes to a marketplace's ontology were sometimes avoided 5 because of the implementation difficulties and risks they posed, even if the changes would improve the marketplace once they were properly implemented.

Accordingly, it would be an advancement to provide tools and techniques to reduce such problems by improving the ease and consistency with which marketplace configuration files are generated.

10

### BRIEF SUMMARY OF THE INVENTION

The present invention provides tools and techniques for facilitating electronic commerce by improving the ease and consistency with which marketplace configuration files are generated. The invention may be embodied in methods, in properly configured computer 15 systems, and in properly configured computer storage media such as CD-ROMs or hard disks. The embodiments use or comprise an express structural ontology specification for an electronic commerce marketplace. The structural ontology specification is organized in a predefined format so that it can be parsed by a computer. It is an express and hence human-readable specification rather than being merely implicit in computer program code, and it preferably 20 expressly specifies at least product categories, product generic attributes, and product category attributes.

A method according to the invention automatically parses the structural ontology specification using a computerized tool. The tool's general purpose is that it is also capable of parsing other structural ontology specifications written in the predefined format; these may 25 specify the ontology of other marketplaces and/or the ontology of different versions of the marketplace of interest. The method extracts ontology information from the structural ontology specification using a computerized tool which can also extract ontology information from other structural ontology specifications. The method uses ontology information extracted from the structural ontology specification to automatically generate for the electronic commerce 30 marketplace configuration materials, at least some of which configuration materials are not product catalog files. Inventive systems and configured storage media similarly use ontology information extracted from an express structural ontology specification to automatically generate configuration materials for an electronic commerce marketplace.

In some embodiments, the structural ontology specification expressly specifies one or more of: a data type for at least one attribute; a data size for at least one attribute; allowed data values for at least one attribute; a search type for at least one attribute; at least one mandatory attribute; at least one optional attribute; and a mapping between an attribute and a product relational database field.

In some embodiments, ontology information from the structural ontology specification is used to automatically generate at least one of: a user interface configuration file such as a user interface search configuration file or a user interface product display configuration file; a search interface configuration file such as a search interface initial database request configuration file or a search interface product detail request interface configuration file; a product catalog configuration file such as a catalog mapping sheet configuration file or a catalog enumeration load sheet configuration file; a file containing a user interface quality assurance checklist; a file containing a search interface quality assurance checklist; a file containing a framework of a script for extracting product data from a text file; a file containing a framework of a script for loading product data into a product relational database; a file containing a product data quality assurance script; a configuration file for a graphical product data entry tool which accepts product data entered manually by a user and places the product data in a product relational database; and a file containing documentation which describes product data that is requested from a supplier regarding products to be offered in the electronic commerce marketplace.

Computer-readable storage media embodiments are properly configured when they contain program code to perform a method of the invention. A computer system embodiment comprises a processor and a memory accessible to the processor. The memory stores (and is thus configured by) an express structural ontology specification for a particular electronic commerce marketplace. The system is further configured by software for the processor which, when executed, uses ontology information read from the structural ontology specification to generate configuration materials for the electronic commerce marketplace.

The express structural ontology specification preferably serves as the authoritative source of ontological information for the electronic commerce marketplace. For instance, in a web site development environment, a discrepancy in category names used in different configuration files can be resolved by referring to and relying on the category name(s) used in the express structural ontology specification. Other features and advantages of the present invention will become more fully apparent through the following description.

## BRIEF DESCRIPTION OF THE DRAWINGS

To illustrate the manner in which the advantages and features of the invention are obtained, a more particular description of the invention will be given with reference to the attached drawings. These drawings only illustrate selected aspects of the invention and thus do not limit the invention's scope. In the drawings:

Figure 1 is a diagram illustrating a prior art approach to commercial web site creation and maintenance, in which product data from a single vendor's catalog is entered manually for use by interactive tools that produce the web site, and the structural ontology of the web site is implicit in the internal code and data of such tools.

Figure 2 is a diagram illustrating another prior art approach to commercial web site creation and maintenance, in which site ontology is somewhat more subject to control, in that several vendors agree on a structural ontology for the site, and yet use is not made of an express structural ontology specification to automatically generate configuration files for the site.

Figure 3 is a diagram illustrating prior art systems and methods that focus on transactional ontology rather than structural ontology.

Figure 4 is a diagram illustrating tools and techniques according to the present invention for using an express structural ontology specification to generate configuration files for at least one marketplace, and then configuring the at least one marketplace accordingly.

Figure 5 is a flow chart illustrating methods of the present invention for using an express structural ontology specification to generate marketplace configuration files.

Figure 6 is a flow chart further illustrating embodiments of an ontology information extraction step shown in Figure 5.

Figure 7 is a flow chart further illustrating embodiments of a configuration file generation step shown in Figure 5.

Figure 8 is a flow diagram illustrating use of an express structural ontology specification according to the invention to facilitate configuration of an operational marketplace.

Figure 9 is a flow diagram illustrating data tools, product data load scripts, and databases in a marketplace architecture configured according to the invention.

Figure 10 is a flow diagram illustrating procurement applications, enterprise resource planning systems, and a product catalog system in a marketplace architecture configured according to the invention.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides tools and techniques for facilitating electronic commerce marketplaces by automatically generating finished and/or template configuration files for a given marketplace from an express structural ontology specification. Although this Detailed Description is organized for convenience and clarity into several sections, it should be read as a whole, with attention to earlier or later portions as needed to aid understanding. The same holds true of the entire current application.

### Incorporation by Reference

This application claims priority to and incorporates by reference the following United States provisional applications: serial no. 60/274,595 filed March 10, 2001, serial no. 60/278,558 filed March 23, 2001, and serial no. 60/280,196 filed March 30, 2001. For clarity and reader convenience, some of the material from incorporated provisional applications is expressly repeated herein, while other material (such as some multi-page script source code) is not so repeated but is included instead through incorporation by reference.

### Overview

As an introductory example, Figure 4 illustrates embodiments of the present invention in which several marketplaces A, B, and C each have a respective structural ontology 400, 402, 404. As indicated, the invention focuses on structural ontology rather than transactional ontology. In particular, an express structural ontology specification 408, 410, 412 exists for each respective structural ontology 400, 402, 404. A given express structural ontology specification 406 may be the result of negotiations between marketplace vendors, as suggested in Figure 2, and need not be automatically generated. However, the specification 406 must be express, not merely implicit in computer programs' internal code and data as is the case with approaches like that shown in Figure 1.

As confirmed by Figure 4, the invention provides tools 414 and techniques for facilitating multiple electronic commerce marketplaces such as marketplaces A, B, and C by automatically generating configuration materials 416, 418, 420 for the respective marketplaces. The invention may alternately or additionally be used to generate 414, from the different versions over time of a single marketplace express structural ontology specification 406, corresponding versions over time of configuration materials for that marketplace. Generated configuration materials may include complete or partial search page load files, product detail display load files, catalog files, data tool configuration files, QA checklists, and others

discussed herein. The materials generated 414 from the express structural ontology specification 406 according to the invention are then combined with other items to create the operational marketplace web site, documents, configured tools, and other elements of a configured marketplace 422, 424, 426. For instance, load files produced with the invention 5 may be combined with product data and conventional database software to populate a product database. Document templates produced with the invention may be tailored to specific suppliers by adding names and addresses. Data tool configuration files produced with the invention may be used to configure data tools such as a catalog transfer manager.

As confirmed in Figures 5-7, embodiments of the invention can facilitate a plurality of 10 electronic commerce marketplaces and/or marketplace versions; the illustrated embodiments include or perform an obtaining step 500, a parsing step 502, an extracting step 504, and a generating step 506. The invention provides tools for performing these steps, such as configured computer systems and configured computer media.

During the obtaining step 500, a system for performing the method is provided with an 15 express structural ontology specification 406 for a particular electronic commerce marketplace. A marketplace may be one of several coexisting marketplaces, or it may be a particular version of an evolving marketplace, or both. The structural ontology specification 406 expressly specifies ontology information such as product categories, product generic attributes, and product category attributes. One example of rules for implementing an express structural 20 ontology specification 406 in a spreadsheet format 804 is shown below, beginning with the column headings:

Database	Att_	Node_	Print	Family	Search	Field
Category	Def_	Id	File	Attribute	Method	Name
Name	Id					
30	Data	PIMS	Req'd?	Website	Allowed	
	Type	Mapping	Y or N	Category	values	
				Name	(for ENUMs)	

In particular:

- Database Category Name - Shows up on web-site exactly as entered. Be sure all cells below category name are clear.
- Prod\_Att\_def\_id - Assigned by PIMS Administrator when data is ready to be pushed (make sure all cells are clear)
- Default\_node\_id - Assigned by PIMS Administrator when data is ready to be pushed (make sure all cells are clear)

	Print_File	- Only on row with Category filled in. Names load files. All capitals, no blanks, prefer 8 characters or less.
	Family Attribute	- Shows up on web-site exactly as entered. No constraints except 50 characters max.
5	Search Method	- Four options: textbox, drop-down, radio, or none.
	Field Name	- Truncated version of Attribute name, becomes field name in PIMS' Oracle database. No capitals, no blanks (use underscore), no slashes, no special characters such as # (use text only).
10	Data Type	- Three basic entries: boolean-char(1) for radio buttons (any Y/N attribute); enum-varchar2(x) where x = 50, 250, 500, or 1000 characters; number(x.y)-varchar2(z) where x is total number of digits y is digits to right of decimal point z = 50, 250, 500, or 1000 characters
15	PIMS Mapping	- Always use PROD_GAyyXz where yy = field size (1,50, 250, 500, or 1000 characters) z = a sequence number that increments by one for each value of the same size
	To avoid conflicts with standard Chemdex PROD_GA numbers, always start the PROD_GA numbers at the following starting values (for each Database Category Name):	
	PROD_GA1X7 PROD_GA50X1 PROD_GA250X3 PROD_GA500X5	
20	Required (Y or N)	- Unless absolutely needed, use N (otherwise, may not be able to load products if an attribute is not given by the supplier).
	Website Category Name	- Always leave blank.
25	Allowed Values (ENUMS)-	Always leave blank for textbox searches. Always leave blank for boolean radio boxes. No leading or trailing spaces, no &. Capitalize the first letter of each word (but maintain proper nomenclature, e.g., NEMA, pH, mA) As last step, put all ENUMs on one line and separate by semicolon followed by a space (e.g., Enum1; Enum2; Enum3; Enum4).
30	Other rules may also be used; indeed, a somewhat different set of rules for completing a specification 406 is given later in this application. Regardless, the specification 406 should be organized in a predefined format so that it can be parsed 502 by a suitably programmed computer; familiar parsing tools and techniques such as those used in parsing computer source code may be readily adapted for use in parsing specifications 406 according to the present invention. Parsing 502 proceeds according to the syntax (format) of the specification 406.	
35	When a particular piece of ontology information such as a product attribute is located during a pass through the specification 406, the value given for that piece of information is extracted 504 by being copied to another memory location by the parsing and extraction software. The software which automatically parses 502 the specification 406 and extracts 504 ontology information from it can also parse other structural ontology specifications written in the predefined format and similarly extract ontology information values from them. The structural ontology specification 406 is an express specification rather than being merely implicit in computer program code. In addition to being in a format that can be parsed 502 by software,	
40		

this means that the specification 406 is human-readable. Indeed, it is preferably comprehensible to people who are not skilled as computer programmers.

In some embodiments the parsing step or tool 502 performs quality assurance tests on the submitted specification 406, either as a precursor to the extraction 504 pass or in a manner interleaved with extraction 504 operations. Some quality assurance tests may also be conducted “manually”, that is, by visual inspection, since the specification 406 is human-readable. Some of the mistakes in an express ontology specification 406 which can be detected manually and/or by a QA script or other QA software include: repeated or missing digit in a default node ID (e.g., 11181 instead of correct value 1181); use of dash instead of correct underscore, or vice versa (e.g. attribute identified as “meter-size” instead of correct “meter\_size”); wrong searchability specified (e.g., “drop-down” instead of “text box” for a given attribute); wrong data size specified (e.g., “varchar2(250)” instead of “varchar2(50)"); blank lines at end of ontology spreadsheet file contained invisible cell reference which should not be there because the cells should instead be clear; category(ies) missing. PIMS mapping, search, and data types can also be checked.

In one embodiment, an ontology specification 406 QA script checks for the following conditions before the invention attempts to generate configuration files from the express specification 406, and provides error or warning messages accordingly:

File Format

Error if blank lines exist within the file  
Error if the # of columns is not correct (more or less = bad)  
Error if there are more header lines than are expected

Data types and sizes

Error if node\_id is not a number (of X digits)  
Error if prod\_att\_def\_id is not a number (of X digits)  
Warn if Database Family Name entries are more than 50 characters long.

Uniqueness

Error if node\_id is not unique within the file  
Warn if prod\_att\_def\_id is not unique within the file  
Error if entries in PIMS Mapping field are not unique within a family  
Error if entries in the Family Name field are not unique within a family

Allowed values checks

Error if entries in Required field are not the allowed values (T, F)  
Error if entries in Search Method field are not the allowed values  
Error if PrintFile field contains spaces, special characters other than \_,

lowercase letters.

Error if PIMS Mapping field entries are not in the right format.

Error if entries in PIMS Mapping field use restricted values

Error if entries in PIMS Mapping field have values beyond what is allowed

5 (PROD\_GA 50X5000, PROD\_GA5000X2)

Once the ontology specification 406 is checked and corrected, it is assumed to be in its final state (for that version of the ontology), with the possible exception of changes to enum values. In subsequent versions of the ontology specification 406, fields may be added, for 10 instance, to meet the requirements of a newer version of a catalog system 822. Other changes to an express structural ontology specification 406 to produce a new version may include, without limitation, removing extra spaces from enums, shortening enums to meet generic attribute name length requirements, spelling corrections in enums and family names, and the addition of attributes for reasons such as refining the classification of products. In one case, an 15 att\_def\_id was added to one column in a spreadsheet embodiment 804 of the express structural ontology specification 406, and node\_id was added in another column, to help create 506 configuration files such as enum files and mapping sheets.

The invention uses ontology information extracted 504 from the structural ontology specification to automatically generate 506 configuration files for the electronic commerce 20 marketplace. Many such configuration files were well known prior to the invention. However, they were generated by hand, or at most in a limited automatic manner that did not use an express structural ontology specification as a basis for creating them. Such prior approaches to configuration file generation failed to provide some or all of the advantages of the present invention, such as the relative ease of preparing revised configuration files to reflect an 25 ontological change, and increased consistency among the configuration materials used by a given marketplace.

Figure 6 illustrates in greater detail both the type of ontology information that may be defined in the format of a given specification 406, and the corresponding extraction 504 steps that can be used to extract that information from the specification 406. Corresponding parsing 30 502 details are not shown but will be understood by those of skill in the art. For instance, the specification 406 may specify data types (e.g., string, integer) for attributes, in which case a corresponding parsing 502 step locates the attribute name and the string or enumeration value that indicates the attribute's data type, and a corresponding extracting step 600 extracts the attribute name and the data type indicator from the specification 406. Likewise, parsing and 35 extracting steps may be used on specifications 406 that specify attribute data size 602, the allowed values of an attribute 604, attribute search type 606, the mandatory 608 or optional

610 nature of a given attribute, and/or a mapping 612 between an attribute and a field in the relational database of product data. A given embodiment of the invention may include zero or more of these elements.

More generally, the method steps and system elements shown in the Figures may be  
5 repeated, omitted, renamed, and/or grouped differently in a particular embodiment, except as required by proper interpretation of the claims granted. Method steps may be performed concurrently and/or in a different order than shown, except to the extent that a result of one step is needed by another step, or to the extent that a particular order is required under a proper interpretation of the claims granted.

10 Figure 7 illustrates in greater detail both the type of configuration materials that may be automatically generated from a given specification 406, and the corresponding generation 506 steps that use extracted 504 ontology information from the specification 406 to create the configuration materials. Corresponding uses of the configuration materials in the final marketplace are not shown in this Figure but will be understood by those of skill in the art. As  
15 with the other Figures, elements may be repeated, omitted, renamed, reordered, and/or grouped differently in different embodiments of the invention. In some embodiments, the method uses ontology information extracted 504 from the structural ontology specification 406 to automatically generate 506 configuration files by one or more of the following automatic generation steps: generating 700 a user interface configuration file such as generating 702 a  
20 user interface product display configuration file and/or generating 704 a user interface search configuration file; generating 706 a search interface configuration file such as generating 708 a search interface initial database request configuration file and/or generating 710 a search interface product detail request configuration file; generating 712 a catalog configuration file such as generating 714 a catalog enumeration load sheet configuration file and/or generating  
25 716 a catalog mapping sheet configuration file; generating 718 a quality assurance checklist such as generating 720 a search interface quality assurance checklist and/or generating 722 a user interface quality assurance checklist; generating 724 a product data quality assurance script; generating 726 a framework of a script for extracting product data from a text file and loading the product data into a product relational database; generating 728 a configuration file for a graphical product data entry tool which accepts product data entered manually by a user and places the product data in a product relational database; and/or generating 730 a file containing documentation which describes product data that is requested from a supplier regarding products to be offered in the electronic commerce marketplace. Computer systems

and/or computer-readable media specifically configured to operate/cause operations according to a method of the invention also embody the invention.

Figures 8 through 10 further illustrate the architecture and context of the invention.

Domain-specific knowledge 800 pertaining to a particular marketplace is captured by and/or from domain experts using an ontology capture tool 802, 414 to produce an express structural ontology specification 804, 406 for the particular marketplace at a particular point in time. The structural ontology specification 804 is express in that it is not embedded in computer code but is instead easily read, understood, and modified by people who are not necessarily trained as computer programmers. The ontology specification 804, 406 is structural as opposed to being transactional, in that it specifies relatively static aspects 800 of a marketplace such as the products, their attributes, and their relationships to one another, rather than specifying more dynamic aspects of the marketplace such as purchase orders 310. In one embodiment, the ontology is captured into a spreadsheet file 804 using a spreadsheet program 802.

Scripts, macros, and/or other code 806, 414 are used to generate 506 configuration materials 808 such as configuration files 700-716, 728, checklists 718-722, scripts 724, script frameworks (partial scripts) 726, and/or documentation 730 for the particular marketplace. These configuration materials 808 are generated 506 from the marketplace's express structural ontology specification 804, 406. The various generated 506 script output files 808 are used to tailor a marketplace technology system 810 having generic components, thereby producing a more fully configured marketplace system 814 having components that are tailored according to the particular structural ontology of the marketplace. Suitable components of the system 810 may include data management tools 818; a catalog system 822; Quality Assurance (QA) tools 824 such as user interface QA checklists 722, search interface QA checklists 720, and QA scripts 724; and a procurement system 820 having user and search engine interfaces. Such components are generic when they have not yet been configured to match the ontology specification 406.

After being configured according to the specified ontology, the catalog system 822 has product attributes and categories according to the specified structural ontology but it does not necessarily yet have data for particular products. The configured procurement system 820 likewise has formats for search pages 1006 and product detail display pages 1008 but does not necessarily yet have access to a complete database 920 of product data. The configuration files 808 are generated 506 automatically with the invention, and then installed (implemented) by familiar operations in the generic marketplace system 810 in order to produce the configured marketplace system 814.

Automatically generated files 808 are also used by marketplace business units 812 to facilitate marketplace business operations. For instance, in some embodiments output files 808 generated 726, 724 from the ontology specification 406 are used by a content engineering department for data extraction 828 and QA 826. A QA script 828 can be generated 506 from the ontology specification 406 to check the user interface against the ontology. Such checks may identify, e.g., categories or families in which attributes are missing, and/or attributes (e.g., drop-down lists) that are missing enums or list entries. The result of the check can be documented in a spreadsheet or other file produced by the QA script. Output files 808 generated 730 from the ontology specification 406 may also be used by the supplier relations department in creating data acquisition collateral 830 such as documents which help suppliers understand more clearly what data about their products is needed for the catalog 822.

These more fully configured marketplace operational processes 816 and the fully configured marketplace technology system 814 are populated with product data to produce an electronic commerce marketplace 832 which is configured with product data in the specified ontology. This need not be a production marketplace, but can be instead a staging (testbed) marketplace which uses “maintenance” search collections 908 and product data 906, and which is then released for use -- after appropriate testing and corrections/revisions to include operational search collections 922 and product data 920 -- as an operational marketplace 834. In some embodiments, the search collections 908, 922 and databases 906, 920 are part of the PIMS (Product Information Management System) system 918, which is a catalog system that was apparently used by Ventro and its clients before the earliest of the incorporated provisional application filing dates.

Note that parts of this overall process for configuring a marketplace with files 808 generated (at least in part) from the ontology specification 406 may be repeated in response to changes in that ontology specification 406. For instance, after the specification 406 is changed, the scripts 806 can be run on it once again, to generate new configuration files 808 which are then installed in the system 810. Facilitating such reconfigurations is an advantage of the invention.

In addition to points made above, the following may be noted in connection with Figure 9. A web catalog manager 900, a contract price manager 902, and a catalog transfer manager 904 are examples of data tools 818. These tools 900, 902, 904 may share a single user interface. The web catalog manager data tool 900 provides an HTML form for product data entry into a product database 906, e.g., an Oracle-brand relational database which is organized according to a database schema. The web catalog manager 900 normally depends on the

ontology, and hence it should be configured (or reconfigured) to reflect the specification 406. The contract price manager data tool 902 helps track buyer-specific pricing information, promotional offers, volume discounts and the like; it generally does not depend on the ontology specification 406. The catalog transfer manager data tool 904 assists in transferring product data from flat files, through FTP and the like into the product database 906; it may depend on the ontology, or it may deal only with database fields that are specified in files 910 provided by a supplier to be uploaded into the database 906.

5 Data may also be loaded into the database 906 by load scripts 828 from a DBA department. The data loading scripts 828 may be provided by a product database administrator 10 or product database administration department, in cooperation with a content engineering department 912. Load sheets may be generated 726 to reflect the ontology specification 406. That is, the express structural ontology specification 406 may be used as input to automatically produce 506 Perl modules (or frameworks thereof) for product data parsing and to create 726 load files for loading product database content into the catalog system 822. The content 15 engineering department 912 accepts raw product and price data 914 from suppliers, and processes it to produce formatted files 916 for uploading product data into the database 906. Such processing may include “physical data normalization” which moves data from PDF, word processor, and other software application-specific file formats into a shared file format; such physical normalization does not generally depend on the ontology specification 406. Content 20 engineering may also perform “logical data normalization” which conforms the data with the marketplace ontology, both by manual means and by automation tools 414. Content engineering may also perform “data quality assurance”, both manual and automated, such as using the QA tools 826.

The express structural ontology specification 804, 406 is preferably used according to 25 the invention to automatically generate 506 ontology-specific files which configure the following to match the marketplace’s specified ontology: web catalog manager 900, load scripts 828, logical data normalization scripts/script frameworks for content engineering 912, documentation 830 which tells suppliers what raw product data is needed from them, the product maintenance database 906, and the production (a.k.a. “operational”) database(s) 922. 30 However, not every one of these need be configured in every instance or every embodiment.

In addition to points made above, the following may be noted in connection with Figure 10. In operation, the procurement application 820 performs various functions in various versions (e.g., in the commercially used Tradex-brand application), such as providing help pages, providing a user interface for other applications in addition to the procurement

application, performing order status and fulfillment tracking, performing other order management functions, supporting a shopping cart, supporting designation of company favorite products and/or individual user favorites, supporting workflow and approvals management, supporting various user preferences or profiles, and login/authentication operations. These  
5 functions are generally not tailored according to the marketplace ontology specification 406.

However, the procurement application 820 also presents a user with search functions through a search user interface 1006. Searches and their results (as displayed to the user) may be tailored according to the ontology specification 406. For instance, a search may specify a product category. Search criteria entered in the search user interface 1006 by the user are used  
10 by the procurement software 820 to generate a corresponding search request 1010 to the database 920 in SQL or another database language. The search request results are then displayed to the user through a product display 1008 in the user interface. If more information about a given product category or individual product is desired, the user may narrow the search using a product detail portion of the user interface 1006, which is used to generate a  
15 corresponding product detail search request 1012, whose results are displayed to the user through a product detail portion 1008 of the user interface.

A buyer procurement application 1022 may be used, such as one of the applications provided by Ariba, CommerceOne, I2, ERP Systems, or other vendors. Such applications 1022 provide functions such as order generation (including search), order management, order status  
20 and fulfillment tracking, user management, user authentication, workflow and approval tracking, accounts receivable, accounts payable, general ledger, and so forth. These buyer procurement applications 1022 are generally not configured according to the invention to reflect the ontology of the marketplace.

A marketplace ERP (Enterprise Resource Planning) system 1014 with one or more  
25 Application Program Interfaces 1016 (e.g., the Chemdex API “CAPI”, a Dot Connect-brand API, and a MarketLink-brand API) communicates as indicated in Figure 10 with the procurement applications 820, 1022. The ERP system 1014 provides functions such as accounts receivable, accounts payable, general ledger, order information management, vendor information master management, and buyer information master management. The ERP system  
30 1014 may include ERP software 308. The ERP system 1014 may communicate as indicated with a marketplace data warehouse 1018 used for business intelligence reporting, and with various supplier transaction systems 1020 for order entry, fulfillment, product warehouse management, and billing/financing. The ERP system 1014, marketplace data warehouse 1018,

and supplier transaction systems 1020 are generally not configured according to the invention to reflect the ontology of the marketplace.

As indicated in Figure 10, the procurement applications 820, 1022 communicate with the marketplace product catalog system 822; suitable catalog systems include at least some PIMS systems 918. The catalog system 822 provides features such as functionality for a product catalog/data repository, pricing lists, pricing based on volumes and/or times, buyer-specific pricing, contract information, vendor information, vendor rankings, and/or search capabilities. The catalog system 822 includes APIs 1002, such as PIMS or MarketLink APIs. It also includes a database management system 920, such as one using commercially available Oracle or Verity software and conventional disks, RAID, or other storage hardware. The product database 920 is organized according to a database schema which is not necessarily marketplace-specific. The product catalog, by contrast, is organized according to attributes and other metadata 1004 based on the marketplace's specified ontology 406. For instance, catalog metadata 1004 may specify product categories, attributes, allowed data values, data types, data sizes, and so on. Conceptually, the metadata 1004 appears in the front half of the catalog; the back half includes the database 920 and its attendant database query language and database fields.

The express structural ontology specification 406 is preferably used according to the invention to automatically generate 506 ontology-specific files which configure the following to match the marketplace's specified ontology: user interfaces for searching 1006 and product display 1008, search interfaces for the initial request 1010 and the product detail request 1012, and the product catalog front half 1004 (as opposed to the catalog's internal back half, which interfaces with the product database 920). However, not every one of these need be configured in every instance or every embodiment.

25

### **Additional Examples, Definitions, and Their Use**

In describing embodiments of invention, the meaning of several important terms is clarified by express definitions and/or by examples, so the claims must be read with careful attention to these clarifications. Specific examples are given herein to illustrate aspects of the invention, but those of skill in the relevant art(s) will understand that other examples may also fall within the meaning of the terms used, and hence within the scope of one or more claims. Important terms may be defined, either explicitly or implicitly, here in the Detailed Description and/or elsewhere in the application file(s). The invention may be embodied in various ways, and it may also be described in various ways. Accordingly, the examples discussed throughout

this document are not necessarily consistent with one another in every particular; rather than omitting details to achieve complete consistency, details are included if they may assist in a better understanding of some embodiment of the invention.

The following terms are preferably used in the indicated manner in at least some embodiments of the invention. The information below includes both examples and definitions; those of skill will understand whether particular details can or must be omitted from, or varied within, a given embodiment. Note that the terms “marketplace” and “vertical” may be used interchangeably.

ATTRIBUTES: One element of an ONTOLOGY. Attributes are aspects of a product which are distinguishing, important, descriptive, and/or informative. There are generic attributes, which every product in a marketplace will have (such as name or catalog number), and there are family-specific (or category-specific attributes) which only products in the same family will share (such as a “Composer Name” or “ISBN Number”). Attributes are assigned to a product according to the product’s CATEGORY, and are mapped onto the product in the database via the ATT\_DEF\_ID. Attributes themselves have attributes or characteristics which are important to capture in an ontology, such as if the attribute is required or not, if it’s searchable, how it’s searchable, its data type and size, its allowed values, etc.

AWK: A pattern scanning and processing language utility found on many UNIX systems. AWK may be used with stream editor SED to create or modify scripts according to the invention in Perl or other scripting languages. SED and AWK are widely used and understood, and are not, in and of themselves, claimed as part of the invention although they may be used to help implement parts of the invention. Likewise, although Perl is used in many of the examples given here, other scripting languages, and other programming languages, may be used in other embodiments of the invention. Scripting languages and their interpreters are widely used for various purposes, but their use according to the present invention is believed to be new. They are not, in and of themselves, claimed as the present invention.

B20: This stands for “build 20” and refers to a specific version of the Ventro front-end UI and procurement system. The use of “B20-compatible” indicates that that entity is usable with (or to be configured with) that version B20 of the front-end user interface and procurement system.

CATEGORIES: One element of an ONTOLOGY. The product space sold by a marketplace is divided up into CATEGORIES, and every product sold by the marketplace is assigned to map into one of these categories. Categories are displayed as the result of certain kinds of searches on the procurement system’s user interface. Also, the category assignment of

a product will control which attributes are assigned to that product. Products are mapped into categories in the database via the NODE\_ID.

CES: Content Engineering Services. Usually refers to the Ventro department which works with Marketplace customers doing catalog and content consulting/training. Can 5 occasionally refer to the marketplace's department which produces catalog and content.

CONFIG FILES, CONFIGURATION FILES: In a narrow sense, each of these phrases refers to files needed to configure PIMS (or other catalog systems) and data tools for a given marketplace, to configure the procurement systems search interface and product display screens, to display search results from a vertical's search engine, and to create DBA load files.

10 Examples include PIMS MAPPING SHEETS, PIMS ENUM LOAD SHEETS, UI CONFIG FILES, UI LOAD SHEETS FOR SEARCH PAGES, UI LOAD SHEETS FOR PRODUCT DETAIL DISPLAY, SUPPLIER TOOLS UI CONFIG FILES. In a broader sense, configuration files are files that are automatically generated – in part or in their entirety – from the ONTOLOGY SPEC SHEET according to the invention. In this broader sense, config files 15 (also called “configuration materials”) include quality assurance checklists and scripts, and documentation for suppliers, generated according to the invention and noted at steps 718, 720, 722, 724, and 730 in Figure 7.

Step 730 may be implemented in a manner similar to the implementation of step 718. See also the checklist.xls and report.txt files noted elsewhere in the application(s), as they are 20 likewise instructional text documents, for end-user use, that are generated 506 from an ONTOLOGY SPEC SHEET 406.

CONFIG FILE GENERATION SCRIPTS: Perl scripts which automatically generate CONFIG FILES and other ONTOLOGY-specific output from an instance of the ONTOLOGY SPEC SHEET. Other scripting languages could also be used, as well as programs written in 25 languages such as C, C++, Java, etc. These scripts are also known as vertical buildout scripts, buildout scripts, and vertical generation scripts.

CVS: The “Concurrent Versions System”. CVS is the version control system used at Ventro to house and manage source code and other configuration files for the Ventro platform and a marketplace's specific set of source code. CVS is generally similar to the RCS and SCCS 30 version control systems. Version control systems are widely used and understood, and are not, in and of themselves, claimed as part of the invention although they may be used to facilitate embodiments of the invention.

DBA: The database administration group, or sometimes an individual database administrator.

DBA LOAD SCRIPTS: A set of SQL and ProC modules used to load ASCII data files into PIMS. These modules are specific for each CATEGORY of product, so a given marketplace will have many modules which need to be produced, based on the ONTOLOGY.

5 ENUM: An enumeration data type indicating an ATTRIBUTE which can only be populated with certain predetermined, allowed values (e.g., the attribute eye\_color is an enum with values blue, green, grey, brown, other). Alternately, one member of a set of allowed values assigned to an ATTRIBUTE (e.g., the value grey is one of the ENUMS of the eye\_color attribute). ENUMS are stored in the enum table in PIMS where a counting number (1,2,3...) is bound to each specific attribute value.

10 FAMILIES: Used interchangeably with CATEGORIES.

FE group: This refers to the “front-end group”, which is a group of developers who oversee creation and customization of the user interface and procurement system.

15 GA: A generic attribute in Ventro’s Product Information Management System (PIMS - the Ventro catalog system) which is configured by a marketplace’s ontology to hold a specific product attribute. The attribute assigned to a given GA will vary by product family.

NODE\_ID: A database value present in the PIMS product table and in the prod\_tree\_def table, which is used to map a product to a CATEGORY.

20 ONTOLOGY: The information about how a marketplace will organize and portray products on its site and in its search engine, and how data in the marketplace’s PIMS database will be stored and configured. Information in an ONTOLOGY may include category assignments, product attributes, and may also include information about the data type, data size, allowed values, searchability, and mandatory or optional nature of each product attribute. An ONTOLOGY may also include NODE\_IDs, PROD\_ATT\_DEF\_ID’s and database field mappings to each CATEGORY’s ATTRIBUTES.

25 ONTOLOGY SPEC SHEET: This is a specification sheet, that is an express description of the structural ontology in a standardized format (so it can be parsed and processed by the CONFIG FILE GENERATION SCRIPTS), which specifies the ONTOLOGY that is to be configured in a vertical’s PIMS instance and presented through a vertical’s search engine. The spec sheet has been implemented as an Excel spreadsheet, but could be implemented as a flat 30 text file, as a sequence of database records, as an XML file, via a web-based user interface which generates one of those file-types, or in other ways.

PIMS: Product Information Management System. Ventro’s Oracle-database-system-based catalog system, search services, and API’s. Used as the product data repository and search systems for Ventro and its marketplaces. One of PIMS’ major components is the

product database; another is the product catalog built on top of the database. Each marketplace has its own installation of PIMS or another catalog system, customized for that marketplace based on the marketplace's ONTOLOGY. The invention is not limited to uses with PIMS.

5 PIMS ENUM LOAD SHEETS: Refers to the generated 712, 728 file ("enumFile") used by the DBA group to load a marketplace's ENUM values into their PIMS instance.

PIMS MAPPING SHEETS: Refers to the files generated by the CONFIG FILE GENERATION SCRIPTS in the "family\_Attribute" directory. They are used by the DBA group to generate DBA LOAD SCRIPTS and for other PIMS configuration tasks.

10 PRINTFILE: A Perl module used by a marketplace's CES team as part of scripted data processing. The PRINTFILE modules write the output of a data processing script to a file in the format necessary for it to be loaded into PIMS using the DBA load scripts. These modules are specific for each CATEGORY of product, so a given marketplace will have many modules which need to be produced, based on the ONTOLOGY.

15 PROD\_ATT\_DEF\_ID: A database value present in the PIMS product table and in the prod\_att\_def table, which is used to map a product to a set of ATTRIBUTES. In one embodiment, the order of the entries in the prod\_att\_def and the prod\_ga files are different; each family will have different sets of PROD\_GA's. Perl scripts use the ontology specification file 406 as a template to generate an auto.sh file, which is edited to dynamically generate loading scripts.

20 PRODUCT TREE: Refers to the prod\_tree\_def table in PIMS. This table contains CATEGORY names and NODE\_ID's.

25 PROPERTIES FILE: A file used by the front-end marketplace development team to configure the procurement system front-end search pages or product detail display pages. There are three types of this file: ontology.properties, enum.properties and radio.properties. An example of an ontology.properties is given later in this application. One embodiment of the invention generates 700-710 an Enum.properties1 file that includes text such as the following excerpt (ellipsis indicates similar omitted material):

```
other=Other
C2.A0.enumType=type
30 C2.A0.segment=1023
C2.A1.enumType=material
C2.A1.segment=1023
C2.A2.enumType=nuts
C2.A2.segment=1023
35 C2.A3.enumType=diameter
C2.A3.segment=1023
C2.A4.enumType=length
```

C2.A4.segment=1023

...

Another example from an enum.properties file is given in incorporated provisional application 60/278,558 at pages 62-63.

5 One embodiment of the invention generates 700-710 from the express specification 406 a Radio.properties file that contains text such as the following; ellipsis again indicates similar omitted material:

```
# The options for the attributes that are radio buttons
# C#= the category ID
10 # A#= the attribute ID
# O#= the option ID
# TTL = the total number of radio options for each attribute
```

```
# Augers-Drilling
```

```
15 # gearbox
C34.A2.O0=yes
C34.A2.V0=T
C34.A2.O1=no
C34.A2.V1=F
20 C34.A2.O2=don't care
C34.A2.V2=@null
C34.A2.TTL=3
```

```
...
```

```
# Terminal Blocks
```

```
25 # other_awg
C220.A18.O0=yes
C220.A18.V0=T
C220.A18.O1=no
C220.A18.V1=F
30 C220.A18.O2=don't care
C220.A18.V2=@null
C220.A18.TTL=3
```

```
# Wire and Cable
```

```
35 # tray_rated
C223.A11.O0=yes
C223.A11.V0=T
C223.A11.O1=no
C223.A11.V1=F
40 C223.A11.O2=don't care
C223.A11.V2=@null
C223.A11.TTL=3
```

In a more recent version of the front end application, the display page is autogenerated

45 from the contents of the database, so the ontology.properties, enum.properties and radio.properties configuration files are used differently.

Notation such as “700-710” indicates that one or more of the listed components 700 through 710 is present in the embodiment(s) being discussed. It does not mean that all of the listed components (e.g., 700, 702, 704, 706, 708, and 710) are present in a given embodiment, although they may be. Nor does it mean that every embodiment of the invention must include 5 one or more such components.

QA: Quality Assurance. This is sometimes used generally, and sometimes used specifically with regard to the ONTOLOGY SPEC SHEET. With regard to the ontology spec sheet (a.k.a., “express structural ontology specification”), quality assurance may include manually checking the sheet for consistency with the vertical’s desired design and using 10 software to check the sheet for violation(s) of unique constraint(s), formatting error(s), naming convention non-compliance, and/or other problems. Error or warning messages can be produced when the following are present in a spec sheet: re-used PROD\_ATT\_DEF\_IDS (should generate a warning message when the config scripts are run, so that a content engineering person can check that case with the marketplace domain experts); use of dash “-” 15 instead of underscore “\_” in attribute name; character array declared larger than desired; use of invisible cell references in blank lines at end of spreadsheet; errors in file format, data types and sizes, uniqueness, or allowed values; CATEGORIES or FAMILIES with missing ATTRIBUTES, or attributes that have missing ENUMS or list entries.

SED: A stream editor found on many UNIX systems; see also AWK.

20 SUPPLIER TOOLS UI CONFIG FILES: Ventro provides marketplaces with web-based data management tools: Web Catalog Manager is a graphical way to enter data into PIMS, and Catalog Transfer Manager allows flat-file upload of data into PIMS. These tools must be configured according to a marketplace’s ONTOLOGY. This can be done using the output of the CONFIG FILE GENERATION SCRIPTS to automatically configure these data 25 tools as required.

UI CONFIG FILES: See PROPERTIES FILE. This is a general way to refer to the set of all PROPERTIES FILES.

30 UI LOAD SHEETS FOR PRODUCT DETAIL DISPLAY: See PROPERTIES FILE. This is the file used by the procurement system user interface developers to configure the fields which appear on a products detail display page in the procurement system. These depend on a marketplace’s ONTOLOGY.

UI LOAD SHEETS FOR SEARCH PAGES: See PROPERTIES FILE. This is the file used by the procurement system user interface developers to configure the category-specific search pages in the procurement system. These depend on a marketplace’s ONTOLOGY.

UI VALIDATION FILE: A text file containing the checklist of items which must be QA'ed in the procurement system user interface and in the marketplace's search functionality. This QA can be done manually or via scripts. This checklist is generated by the CONFIG FILE GENERATION SCRIPTS and is based on the marketplace's ONTOLOGY.

5

### Comments on Computers

Computers may be configured for use as tools 414, as components of a configured marketplace 814, and otherwise according to the invention. In general, it will be understood that scripts, applications, databases, files, interfaces, and the like refer to computers in that they run on computers, reside on computers, and provide control over computers. Suitable computers may be one or more of a workstation, a laptop computer, a disconnectable mobile computer, a server, an embedded system, a mainframe, or a handheld computer, for instance. A given computer may be a general purpose computer configured by software (e.g., scripts) according to the invention, or it may be a special purpose machine configured by ASICs, FPGAs, or the like. A processor may be a uniprocessor or a multiprocessor component of the computer. A suitable computer system often includes one or more user I/O devices such as a display screen, keyboard, mouse, microphone, speakers, touch screen, and so on. The computer system includes random access memory and may include other forms of memory such as ROM or PROM memory. The memory is in operable communication with the processor, and the I/O devices likewise communicate with the processor and/or the memory.

The computer system is capable of using floppy drives, tape drives, optical drives or other means to read a storage medium. A suitable storage medium includes a magnetic, optical, or other computer-readable storage media having a specific physical substrate configuration.

Suitable storage devices include floppy disks, hard disks, tape, CD-ROMs, DVDs, PROMs, RAM and other computer system storage devices. The substrate configuration represents data and instructions which cause the computer system to operate in a specific and predefined manner as described herein. Thus, a given medium tangibly embodies a program, functions, and/or instructions that are executable by the computer system to perform one or more steps for facilitating electronic commerce substantially as described herein.

The computer(s) in a system according to the invention may be connectable to one or more networks through network I/O hardware and/or software. By way of example, suitable computer networks include local networks, wide area networks, and/or the Internet. "Internet" as used herein includes variations such as a private Internet, a secure Internet, a value-added network, a virtual private network, or an intranet. A network may include one or more LANs,

- wide-area networks, Internet servers and clients, intranet servers and clients, or a combination thereof. Such computer networks may form part of a telecommunications network and/or interface with a telecommunications network. The network's transmission media may include twisted pair, optical fiber cables, coaxial cable, telephone lines, satellites, radio waves,  
5 microwave relays, modulated AC power lines, and other data transmission "wires" known to those of skill in the art, as well as routers, bridges, caching appliances, and the like. Note that the term "wire" as used herein includes wired and/or wireless communications. Methods such as TDMA, CDMA, FDMA, and other encoding and/or multiplexing methods may be used, as well as GSM, PDC, Wireless Application Protocol, and other technologies and protocols.  
10 Signals according to the invention may be embodied in volatile and/or nonvolatile network transmission media.

Although particular components are shown in the Figures and/or discussed herein, those of skill in the art will appreciate that the present invention also works with a variety of other networks and computers, as well as a variety of batch files, scripts, tools, and other  
15 software. To avoid repetition, it is assumed here that discussion of an inventive computer system will be applied by those of skill to promote understanding of the inventive methods, and vice versa. Likewise, the discussions of the inventive methods may be applied by those of skill to inventive computer storage media which are configured with software to operate according to the methods.

20

### **Example Ontology Specification Completion Rules**

In one embodiment, instructions for filling in an ontology specification 406 include the following; another such set of rules is given earlier in this application:

#### Filling out the Ontology Specification Sheet

25 The Ontology Specification Sheet (ontology spec sheet) is an Excel spreadsheet that contains information about product categories, category specific attributes, attribute search methods and attribute data types pertaining to a Ventro marketplace.

Buildout scripts run off of the tab-delimited form of this file and generate database configuration files, product load scripts and the front-end properties files needed to create a  
30 Ventro marketplace. The ontology spec sheet is also used to configure Data Tools' Web Catalog Manager and Catalog Transfer Manager.

The ontology spec sheet contains the following columns, in order from left to right:

#### Database Category Name

35 The Database Category Name is the product family or category name that is displayed on the front-end.

The Database Category Name is loaded into the PROD\_TREE\_DEF table in PIMS as the NODE\_NAME using the load file prod\_tree\_def.out which is generated [712, 728] by the

buildout scripts. The prod\_tree\_def.out load file contains the NODE\_NAME/NODE\_ID pairings.

The Database Category Name is included in the ontology.properties file also generated by the buildout scripts. This file is used by the front-end developers to configure the category search pages and the product detail pages.

#### Att\_Def\_Id

The Att\_Def\_Id is an internal reference number, usually four to five digits in length, that is assigned to a set of family or category specific attributes. It must be numerical; no characters are allowed.

The Att\_Def\_Id is usually unique in a marketplace's ontology. The Att\_Def\_Id can be shared by more than one family; however, the set of attributes of each category must be the same.

The Att\_Def\_Id is stored in three locations in the database. It is the primary key of the PROD\_ATT\_DEF table where it is called the PROD\_ATT\_DEF\_ID. It is loaded into this table as part of the prod\_att\_def.out file which is generated [712, 728] by the buildout scripts. The Att\_Def\_Id is also loaded into the PRODUCT table with each new product record as the PROD\_ATT\_DEF\_ID. The last table the Att\_Def\_Id is loaded into is the ENUM table. Here it is known as the PROJECT\_SEGMENT. The load file for the ENUM table, enum\_file.out, is generated by the buildout scripts.

Another file generated by the buildout scripts that utilizes the Att\_Def\_Id is the Enum.properties file. This file, along with the ontology.properties file, is used by front-end developers to configure the category search page for the attributes with a *drop-down* search type.

#### Node\_Id

The Node\_Id is an internal reference number, usually four to five digits in length, that is assigned to a Database Category Name. This number MUST be unique in a marketplace's ontology. It must be numerical; no characters are allowed.

The Node\_Id is stored in two locations in the database. It is the primary key of the PROD\_TREE\_DEF table. The buildout scripts generate [712, 728] a load file, prod\_att\_def.out, which contains the Node\_Id/ Node\_Name pairing. This file is used to load the PROD\_TREE\_DEF table.

The NODE\_ID is also loaded with each product record into the PRODUCT table as the DEFAULT\_NODE\_ID. The DEFAULT\_NODE\_ID maps a product to its respective category name.

#### Print\_File

The Print\_File is an abbreviated version of the database category name and is used to name a variety of load scripts and files.

Print\_File names are used by the buildout scripts to name the all\_tags text files. The product load script, qaTemplate.pl [724], to QA product load files, utilizes the all\_tag files. The all\_tag files contains the data types, data sizes and allowed values for family specific attributes that are accessed by the QA script.

The Print\_File name is also used by the buildout scripts in the load\_setup.out file. The load\_setup.out file is used by the Database Engineers to programmatically generate [726] family specific product load scripts. The Print\_File name along with the supplier short name is used to name family specific product load files.

Because it is used to name files residing in the UNIX operating system, no spaces are allowed. Underscores are used instead. The Print\_File name is always composed of upper-case

letters and cannot be longer than twenty-seven characters in length. Also, the Print\_File name cannot contain any numerals. Please see formatting rules below for further restrictions.

#### Family Attribute

5 Family Attributes are the names of the family or category specific attributes that will be displayed on the front-end on the category search and product detail pages.

The values entered here are included in the ontology.properties file generated by the buildout script. This file is used by the front-end developers to help configure the front-end category search and product detail pages.

10 The names of searchable attributes will be shown on the category search page and the product detail page. Attributes with a search method of *none* will NOT appear on the category search page. Attributes with a search method of *none* will NOT appear on the product detail page unless otherwise specified.

#### Search method

15 The Search method specifies the method used for searching over a family or category specific attribute on the category search page. An attribute can have only one Search method.

20 The types of search method are *text box* (text string), *drop-down* (drop-down list), *radio* (radio button) and *none*. A *text box* search allows the user to use any string of text they enter as their search query. A *drop down* search consists of a predetermined list of allowed values (drop-down list) that can be selected by the user for their search query. A search method of *radio*, as a default, allows the user to enter a yes or no argument as their query. An attribute with a search method of *none* is not searchable.

25 Attributes with a search method of *text-box*, drop-down and radio will appear on the category search page and as a default, on the product details page. Attributes with a search method of *none* will NOT appear on the category search page. Attributes with a search method of *none* will appear on the product detail page only if there is a specific request that they do so.

30 The Search method is tied to the Data Type (see below). Make sure they are compatible. Listed below are the appropriate pairings of Search Methods and Data Types:

#### Search method/Data Type

35 *text box*: *text-varchar2(n)* or *number(y,z)-varchar2(n)* where *n* equals the maximum number of characters or digits allowed in the field, *y* equals the total number of digits and *z* equals the number of digits to the right of the decimal point. The allowed values for *n* are 50, 250, 500 and 1000.

*drop-down*: *enum-varchar2(n)* where *n* equals the maximum number of characters or digits allowed in the field. The allowed values for *n* are 50, 250, 500 and 1000.

*radio*: *boolean-char1*

40 *none*: *varchar2(n)* where *n* equals the maximum number of characters or digits allowed in the field. The allowed values for *n* are 50, 250, 500 and 1000.

#### field\_name

45 The database field name for the family attribute. It is stored in the PROD\_ATT\_DEF table as the primary key ATT\_NAME. The Field Name is an abbreviated version of the family attribute name. It is always in lowercase. No spaces are allowed; underscores are used instead. Please see formatting rules below for more details. The field\_name is incorporated by the ontology buildout scripts into the prod\_att\_def.out load file.

#### Data Type

50 The buildout scripts utilize the Data Type to help create the all\_tags text files. The all\_tags text files are utilized by the QA script, qaTemplate.pl, to check data types, data sizes,

allowed values and whether or not required attributes have a value present in product load file prior to loading.

Allowed values for this field are as follows: *varchar2(n)* for a search method of *none*. *boolean-char1* for a search method of *radio*. *number(y.z)-varchar2(n)* or *text-varchar2(n)* for a search method of *text box*. And *enum-varchar2(n)* for a search method of *drop-down* where *x* equals the maximum number of characters or digits allowed in the field. The allowed values for *n* are 50, 250, 500 and 1000. See *Search method* above.

#### PIMS mapping

The PIMS mapping specifies which column in the PRODUCT or SKU table a family specific attribute will be placed. These columns in the PRODUCT table are known as Generic Attribute or GA columns.

A PIMS mapping is REQUIRED for every attribute listed on the ontology spec sheet.

The format for a PIMS mapping is PROD\_GAnXy or SKU\_GAnXy. PROD stands for the PRODUCT table, SKU stands for the SKU table, *n* is the maximum field size (in bytes or the number of characters/numerals) and *y* is the sequential number of the generic attribute with the same field size in the same family. For example, PROD\_GA500X6 is an attribute with a maximum field size of 500 bytes, and it is the sixth attribute in that family with a 500 byte field size.

There are a limited number of GA columns in the PRODUCT and SKU tables. Each field size has a limited number of columns. In the PRODUCT table there are 20 columns each with a field size of 1, 50 and 250 bytes, 30 columns with a field size of 500 bytes and 10 columns with a field size of 1000 bytes. In the SKU table there are 10 columns with a field size of 50 bytes.

A number of the GA columns are currently set aside for regulatory and shipping related attributes. The GA columns unavailable are given in the formatting rules section listed below.

When sizing columns for attributes, be certain the column is long enough to hold all of the data. A column with a data type of *varchar2(n)* will adjust the length of the column to fit the size of the data (providing its less than the maximum length, *n*). Do not needlessly waste large columns on attributes that will only have small data elements.

PIMS mappings are loaded into the column COL\_NAME in the PROD\_ATT\_DEF table in PIMS using the prod\_att\_def.out file. PIMS mappings are loaded into the column, COLUMN NAME, in the ENUM table in PIMS using the enum\_file.out load file. PIMS mappings are also incorporated by the buildout scripts into the all\_tags text files and the Family\_Attribute Excel files.

#### Required? (Y or N)

This column specifies whether a family specific attribute is required in the database. The allowed values are Y for *required* and N for *not required*.

This information is incorporated by the buildout scripts into the prod\_att\_def.out file. This file is loaded into the PROD\_ATT\_DEF table in the REQUIRED column.

The information in this column is also incorporated into the all\_tags text files. The all\_tags text files are utilized by the QA script, qaTemplate.pl, to check data types, data sizes, allowed values and whether or not required attributes have a value present in product load file prior to loading.

A value of Y or N MUST be given for every attribute.

#### Website Category Name

Obsolete.

Allowed Values

5 Allowed Values are a set of given values for a particular family specific attribute. For an attribute with a search type of *drop-down*, only the values listed here can be loaded into the database. The Allowed Values constrain what values can be used to search over a particular attribute. This is done via a drop-down list on the category search page.

The allowed values for an attribute should be in the order (left to right) that they should appear in the drop-down list (top to bottom). The first value in the drop-down list is the default of *Any*. Do not enter *Any* in the list. This is inserted by the front-end developers later. Separate values by a semi-colon and then a space.

10 The Allowed Values are loaded into the REPRESENTATION column in the ENUM table in PIMS. The load file, enum\_file.out, is generated by the buildout scripts. The other data elements in the enum file are: the PROJECT\_ID, PIMS, assigned by the buildout scripts, the PROJECT\_SEGMENT is the attribute set's PROD\_ATT\_DEF\_ID, the COLUMN\_SEGMENT is the PIMS mapping for the appropriate attribute, and the VALUE is assigned by the buildout scripts. The first Allowed Value has a value of 0, the second a value of 1, and so on.

15

The Allowed Values in the ENUM table will be used to populate the drop-down lists on the category search pages on the front-end.

Ontology Spreadsheet Formatting Rules

## Universal rules

- 1) Double quote marks ("") are NOT allowed anywhere in the ontology spec sheet.
- 2) Ampersands (&) are NOT allowed anywhere in the ontology spec sheet.
- 3) Avoid special characters such as.....
- 25 4) Avoid trailing spaces at the end of all of the fields.
- 5) Empty rows are NOT allowed.
- 6) A marketplace may have only one ontology spec sheet.

## Database Category Name

- 30 1) The value entered here will appear exactly as is on the category search and product return pages on the front-end.
- 2) Can be no longer than 50 characters in length.
- 3) Placed in the left-hand cell, at the top row, of a list of family specific attributes.
- 4) All of the cells between Database Category Names must remain empty.
- 35 5) Single quote marks are NOT allowed.

## Att\_Def\_Id

- 1) Assigned by the marketplace.
- 2) Usually four to five digits in length.
- 40 3) Cannot be longer than 12 digits in length.
- 4) Numeric field only. No letters or punctuation marks allowed..
- 5) In general, this number is unique in a marketplace's ontology.
- 6) This number can only be shared by two categories that have exactly the same attribute set. This includes having the same lists of allowed values in the same order.
- 45 7) Placed on the same row as the Database Category Name. All of the cells between Att\_Def\_Ids must remain empty.

## Node\_Id

- 50 1) Assigned by the marketplace.
- 2) Usually four to five digits in length.
- 3) Cannot be longer than 12 digits in length.

- 4) Numeric field only. No letters or punctuation marks allowed.
- 5) This number is unique in a marketplace's ontology. A Node\_Id cannot be shared by two or more families.
- 6) Placed on the same row as the Database Category Name. All of the cells between  
5 Node\_Ids must remain empty.

**Print\_File**

- 1) Assigned by the marketplace. It's usually an abbreviated form of the Database Category Name.
- 10 2) ALL IN UPPER CASE LETTERS. No lower case characters.
- 3) Can be no longer than 27 characters in length.
- 4) No spaces, commas or other punctuation marks allowed.
- 5) No hyphens.
- 6) Underscores can be used instead of spaces or hyphens.
- 15 7) Numerals are not allowed.
- 8) Placed on the same row as the Database Category Name. All of the cells between Print\_File names must remain empty.
- 9) No trailing spaces.

**Family Attribute**

- 1) The value entered here will appear exactly as is on the category search and product return pages on the front-end.
- 2) Can be no longer than 50 characters in length.
- 3) There must be a value for every family attribute listed in the ontology spec sheet.

**Search method**

- 1) Only one of four options is allowed: text box, drop-down, radio, or none.
- 2) There must be a Search method for every family attribute listed in the ontology spec sheet.

**Field Name**

- 1) Usually is a truncated version of the Family Attribute name.
- 2) Use only lowercase letters. No capital letters are allowed.
- 3) No spaces are allowed. Use underscores instead.
- 35 4) No punctuation marks are allowed, i.e. commas, single quote marks, semi-colons, colons etc.
- 5) No trailing spaces.
- 6) No longer than 50 characters in length.

**Data Type**

- 1) The data type is determined by the search method.
- 2) The basic entries are: varchar2(n), boolean-char(1), enum-varchar2(n), text-varchar2(n) and number(x.y) varchar2(n), where n = 50, 250, 500, or 1000 characters, y equals the total number of digits and z equals the number of digits to the right of the decimal point.
- 45 3) The pairings are:

Search method	Data Type
a) none	varchar2(n)
b) text box	text-varchar2(n) or number(x.y)-varchar2(n)
c) drop-down	enum-varchar2(n)
d) radio	boolean-char(1)

PIMS Mapping

- 1) Use PROD\_GAnXy or SKU\_GAnXy. Where PROD stands for the PRODUCT table; SKU stands for the SKU table. Where  $n$  is the maximum field size (in bytes or the number of characters/numerals). Where  $y$  is the sequential number of generic attributes with the same field size, in the same family.
- 5       $n$  = field size (1,50, 250, 500, or 1000 characters).
- $y$  = a sequence number that increments by one for each value of the same size.
- 10     2) Case is important! The PROD\_GA, SKU\_GA and the X are UPPER CASE LETTERS.
- 15     3) In general, the  $n$  in PIMS mapping and the  $n$  in data type are equal. The  $n$  in data type can be less than the  $n$  in PIMS mapping, but the  $n$  in PIMS mapping can NEVER be less than the  $n$  in data type.
- 4) To avoid conflicts with preassigned GA columns (mentioned above), do not use the following PROD\_GA numbers: PROD\_GA1X1 to PROD\_GA1X6; PROD\_GA1X20; PROD\_GA250X1; PROD\_GA250X2; and PROD\_GA500X1 to 500X4.

Required (Y or N)

- 20    1) The only allowed values are Y for required and N for not required.
- 2) This must be filled in for every family attribute.
- 3) Unless absolutely needed, use N (otherwise, you may not be able to load products if an attribute is not given by the supplier).

25    Website Category Name

Always leave blank.

Allowed Values (ENUMS)

- 30    1) Always leave blank for attributes with a search method of text box.
- 2) Always leave blank for attributes with a search method of radio.
- 3) Always leave blank for attributes with a search method of none.
- 4) No leading or trailing spaces.
- 5) No ampersands (&).
- 35    6) Values should be mixed case with the first letter of each word capitalized (but maintain proper nomenclature, e.g., NEMA, pH, mA).
- 7) Values are separated by a semi-colon and a space (e.g., Enum1; Enum2; Enum3; Enum4).
- 8) Semi-colons are not allowed within a value; otherwise, two values will be created.

40    Sample Express Structural Ontology Specifications

An example of an express structural ontology specification 406 in spreadsheet form 804 is shown in Figures 10A through 10N of incorporated provisional application no. 60/274,595, which show screen shots of sample pages from a such specification spreadsheet 804. For convenience, that example is also summarized textually below.

45    The example is in a file named Promedix\_Ontology\_Spec.xls, in Microsoft Excel spreadsheet file format. It includes a worksheet “Category Specification” with columns entitled “Category”, “Search refinement page”, and “Attribute display page”, but in this instance all

other cells of the worksheet are empty. It also includes a worksheet “Attribute Specification” which has columns entitled “Category”, “Category Expansion”, “Category ID”, “Family”, “Family Attribute Name”, “Search Method”, “Field Name”, “Data Type and Size”, “PIMS mapping”, and “Allowed values (for ENUMS)”. In one instance, the first three columns  
5 contain data such as the following:

	<u>Category</u>	<u>Category Expansion</u>	<u>Category ID</u>
	Bottle	ADHESIVES~Bottle	105640200000
	Electromedical	ADHESIVES~Electromedical	105640300000
	Other	ADHESIVES~Other	105640650000
10	Silicone	ADHESIVES~Silicone	105640725000
	Tapes	ADHESIVES~Tapes	105640750000
	ANTHROPOMETERS	ANTHROPOMETERS	126300000000
	ANTI-NAUSEA PRODUCTS	ANTI-NAUSEA PRODUCTS	131600000000
	ANTI-SLIP MATERIAL	ANTI-SLIP MATERIAL	131960000000
15	ANTI-SNORE COLLARS/MASKS	ANTI-SNORE COLLARS/MASKS	132000000000
	AROMATHERAPY	AROMATHERAPY	143750000000
	ATOMIZER COOLING DEVICE	ATOMIZER COOLING DEVICE	147750000000
	BACTERIA FILTERS	BACTERIA FILTERS	154520000000
	...	...	...
20	FOOT PADS (SEE PADS, FOOT PADS)	APPAREL~Footwear~FOOT PADS APPAREL~Footwear~ FOOT PADS~ (SEE PADS, FOOT PADS)	137600050450 137600050451
	...	...	...

25 Cells in the Family column contained question marks, or values such as “Anesth”, “Apparel”, “Services”, “Lab”; the other cells in the worksheet were apparently empty.

Promedix\_Ontology\_Spec.xls also includes a worksheet “FYI – Generic Attributes” with columns entitled “Search Method”, “Field Name”, “Data Type and Size”, “PIMS mapping”, and “Allowed values (for ENUMS)”. In this instance, the Search Method cells and  
30 Allowed values cells are empty, and the other three columns’ cells contain values such as the following:

	<u>Field Name</u>	<u>Data Type and Size PIMS mapping</u>
	category	char(50)
	name	char(500)
35	quantity	number
	unit_multiplier	number(8)
	units	char(20)
	price	number(10,2)
	description	CLOB
40	synonyms	char(2000)
	applications	char(4000)
	discipline	char(500)
	additional_info	char(4000)
	references	char(4000)
45	seals_of_approval	char(250)

	certificate_of_analysis	char(250)	
	storage_temp	char(500)	
	shipping_temp	char(500)	
	related_products	char(500)	
5	weight	number(15,5)	CMDX internal sequence #, but possibly useful for you to store?
	sku_id	number(12)	
	product_id	number(12)	CMDX internal sequence #, but possibly useful for you to store?
10	cmdx_sku	char(250)	CMDX internal, generated from other fields. Possibly useful for you to store?
	supplier_cat_num	char(40)	The VWR catalog number. Must be unique.
	orig_mfg_catnum	char(64)	Your original manufacturer's catalog number - your vendor cat num field.
15	shipping_type	number(5)	set of fixed values describing how a product is shipped (dry ice, etc). You may want to store.
	review_enum	number(5)	set of fixed values relating to regulatory review and control. You may want to store.
	agency_enum	number(5)	set of fixed values relating to regulatory review and control. You may want to store.
20	regulatory_action_enum	number(5)	set of fixed values relating to regulatory review and control. You may want to store.
	regulatory_schedule_enum	number(5)	set of fixed values relating to regulatory review and control. You may want to store.
25	MSDS	CLOB	text only
	image	BLOB	We accept jpgs and gifs only. Need to be correlated to an individual product.
	state_enum	number(5)	5 for all products not available via Labpoint but are needed in XREF. 4 for all products available in Labpoint.
30	barcode	number(20)	NOT USED CURRENTLY
	expiration_date	date	NOT USED CURRENTLY
	UN_Number	number	NOT USED CURRENTLY
	shipping_weight	numer(15,5)	NOT USED CURRENTLY
35			

Promedix\_Ontology\_Spec.xls also includes a worksheet "CHEMDEX Category Specification" with columns entitled "Category", "Search refinement page", and "Attribute display page", containing values such as those shown below; for clarity the string "Same as current Equipment" used in the spreadsheet is indicated below by "ScE" and the string "Current generic + all family" used and highlighted in the spreadsheet is indicated below by "Cg+af":

	<u>Category</u>	<u>Search refinement page</u>	<u>Attribute display page</u>
	Cleaning and Sterilization Instruments	ScE	ScE
	Accessories and Parts	ScE	ScE
45	Other Instruments and Equipment	ScE	ScE
	Gloves	Cg+af	Cg+af
	Personal Protection	Cg+af	Cg+af

Safety Products	ScE	ScE
Ready-to-Use Gels and Columns	Cg+af	Cg+af
Filters and Membranes	Cg+af	Cg+af
Glass-, Plastic- and other labware	Cg+af	Cg+af
5 Other Lab Supplies	Cg+af	Cg+af
Furniture and Office Supplies	ScE	ScE
Books and Videos	Cg+af	Cg+af
Software	Cg+af	Cg+af
Services	ScE	ScE

10

**8 new search pages    8 new product detail pages**

Promedix\_Ontology\_Spec.xls also includes a worksheet “CHEMDEX Attribute Specification” with columns entitled “Category”, “Family Attribute”, “Search Method”, “Field Name”, “Data Type and Size”, “PIMS mapping”, and “Allowed values (for ENUMS)”. A preface to the worksheet states the following in red letters: “All categories contain the current set of generic attributes as they currently exist (default\_node\_id = 1000), PLUS what's listed below”. The initial Category cells contain values such as “Sensors, Analyzers and pH Meters”, “Spectrometers”, and “Balances”, with corresponding Family Attribute cells containing “NONE” and the corresponding cells of other columns apparently empty. Then a row occurs with the following cell contents:

<u>Category</u>	<u>Family Attribute</u>	<u>Search Method</u>	<u>Field Name</u>
Gloves	Material		material
25	<u>Data Type and Size</u>	<u>PIMS mapping</u>	<u>Allowed values (for ENUMS)</u>
	enum		latex = 1 cotton = 2 neoprene = 3 silicone rubber = 4 butyl synthetic rubber = 5 rubber = 6 vinyl = 7 nitrile = 8 PVA = 9 PVC = 10 viton = 11 hypalon = 12 polyethylene = 13 aramide = 14 EVOH = 15 asbestos = 16 leather = 17 nylon = 18 wool = 19 aluminium = 20 Other = 50
30			
35			
40			
45			

Unspecified = 51

The next two rows contain empty cells except as follows:

	<u>Family Attribute</u>	<u>Search Method</u>	<u>Field Name</u>	<u>Data Type and Size</u>
5	Sterile	radio?	sterile	boolean
	Powder free	radio?	powder_free	boolean

The subsequent rows contain additional values, including Family Attribute cell values such as “Pore size/MWCO”, “Particle size”, “Dimensions”, “Number of wells”, “Column type”, “Compatibility”, “Material”, and “Sterile”; Search Method cell values such as “radio?”, “text box”, and “drop-down”; Field Name cell values such as “pore\_size\_MWCO”, “particle\_size”, “dimensions”, “number\_of\_wells”, “column\_type”, “compatibility”, “material”, and “sterile”; and Data Type and Size cell values such as “char(250)”, “char(500)”, “number(3)”, “boolean”, and “enum”.

15       The preceding is merely one example. Other express structural ontology specifications may differ from this example in various ways. As an example, some columns may be omitted and/or others added. If only one search method is used, no column of cells specifying different search methods would be needed. Field name and Family attribute could be combined into a single column in some specifications. The strings used to represent column names and cell values could vary. The data types and sizes available may vary. Mappings to database fields might be implicit in field names, or might be omitted if the specification is not used to generate data load scripts. Columns may be organized into worksheets differently; another specification includes only a Category Specification worksheet with columns entitled “Category”, “Search refinement page”, and “Attribute display page” plus an Attribute Specification worksheet with 20 columns entitled “Family Attribute”, “Search method”, “Field Name”, “Data Type”, “PIMS mapping”, and “Allowed values (for ENUMS)”. Those of skill in the art will recognize that combinations of these and other variations are possible in different embodiments of the invention, given the teachings of the entire application and the knowledge brought to them by such persons. In one embodiment, the express structural ontology specification 406 includes a 25 “required fields” identification, which is used for instance by the group of people who are responsible for configuring data tools 818 such as the web catalog manager 900. In one embodiment, the specification 406 includes a “printfile” identification, which is used for instance to create 506 print modules and QA modules.

30       Another example of an express structural ontology specification 406 in spreadsheet form 804 is given in incorporated provisional application no. 60/280,196 at pages 49-51. For

convenience, that example is also provided below. This ontology specification 406 was apparently used to generate output files shown in that incorporated application. In text format, the ontology data in the specification 406 includes the following excerpts:

```

CATEGORY prod_att_def_iddefault_node_idprint fileFamily
5 Attribute:Search method Field NameDataType PIMS mapping
    Required? Website Category Name Allowed values (for
ENUMS)
Piping Valve-Ball 1057 1085 PVBALL Type: drop-down
    type enum-varchar2(50) PROD_GA50X1 N One
10 Piece Body; Two Piece Body; Three Piece Body; Three Way Split
Body; Other
    Size (Inches): drop-down size enum-
varchar2(50) PROD_GA50X2 N 0.125; 0.25; 0.375;
0.5; 0.75; 1; 1.25; 1.5; 2; 2.5; 3; 4; 5; 6; 8; 10; 12; 14; 16;
15 18; 20; 24; 30; 36; Other
    Manufacturer: drop-down manufacturer
enum-varchar2(50) PROD_GA50X3 N APOLLO;
ASASHI; ATOMAC; CHEMTROL; COOPER; DIXON; DRAGON; DRESSER;
DURCO; KITZ; KTM; MCCANNA; MILWAUKEE; NELES-JAMESBURY; POWELL;
20 VELAN; VICTAULIC; VOGT; WATTS; WKM; OTHER
    Connection: drop-down connectionenum-
varchar2(50) PROD_GA50X4 N Butt Weld (BW); Flanged
(FLG); Socket Weld (SW); Sweat; Threaded (THD); Threaded X
Socket Weld ; Other
25 ...
    Insulation Type: drop-down
insulation_typeenum-varchar2(50) PROD_GA50X3 N
    Bare; THW; THHN; THWN; XHHW; RHH; RHW; TFFN; TF; TFF; AWM;
MTW/AWM; ACTHH; HCFC; ACT (Bare); SDT/TC; XLP; EPR; FR-PVC;
30 Polyethylene; PVC; Polypropylene; Teflon; Other
    Conductor Groupings:drop-down
    conductor_groupings enum-varchar2(50) PROD_GA50X4 N
        Single; Pair; Triad; Other
35 ...
    "No. of Conductors,Pairs,Triads:" drop-
down number_of_conductorsenum-varchar2(50) PROD_GA50X5 N
    1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 15; 16;
17; 18; 19; 24; 25; 27; 36; 49; 50; 51; 102; Other
40 ...
    Conductor Size:drop-down conductor_size
enum-varchar2(50) PROD_GA50X6 N 22; 20; 18;
16; 14; 12; 10; 8; 6; 4; 2; 1; 1/0; 2/0; 3/0; 4/0; 250; 350;
500; 600; 750; 1000; Other
45 ...
    Color: drop-down color enum-
varchar2(50) PROD_GA50X7 N Black; White; Red;
Blue; Yellow; Green; Orange; Brown; Purple; Gray; Blk/Wht;

```

Blk/Rd; Rd/Blk/Wh; Other

		Strand:	drop-down	strand	enum-
5	varchar2(50)	PROD_GA50X8	N	Solid;	Stranded; Other
		Metallurgy:	drop-down	metallurgy	enum-
	varchar2(50)	PROD_GA50X9	N	Copper;	Aluminum; Other
10	varchar2(50)	PROD_GA50X10	N	Armor	drop-down armor enum-
					None; Aluminum;
		Galvanized Steel; Other			
15	varchar2(50)	PROD_GA50X11	N	Jacket	drop-down jacket enum-
	PVC; Other				None; Chrome PVC; Std
			Tray Rated?	radio	tray_ratedboolean-
	char(1)	PROD_GA1X7N			
		Shield:	drop-down	shield	enum-
20	varchar2(50)	PROD_GA50X12	N	Unshielded;	Overall
		Shield; Individual Shield; Overall and Individual; Other			
		Wire and Cable - Accessories 1325 1353 CABLEACC Type:			
25		drop-down type enum-varchar2(50)	PROD_GA50X1	N	
		Cable Clips; Cable Cutters; Cable Pullers; Cable Tie			
		Wraps; Crimp Tools; Fish Tapes; Heat Shrink Tubing; Pulling			
		Grips; Pulling Line; Spiral Wrap; Electrical Tape; Wire Labels;			
		Wire Lubrication; Wire Strippers; Other			
30	Wireway	1326 1354 WIREWAY	Type:	drop-down type enum-	
	varchar2(50)	PROD_GA50X1	N	Wireway; Slotted	
	Raceway; Adapter; Connector; Cover; Cross; Elbow45; Elbow90;				
	Hanger; Reducer; Tee; Other				
35	varchar2(50)	PROD_GA50X2	N	Material:	drop-down material enum-
				Metal; PVC; Other	
		Width (Inches):	drop-down width	enum-	
40	varchar2(50)	PROD_GA50X3	N	1; 1.5; 2; 3; 4; 6; 8;	
	Other				
		Depth (Inches):	drop-down depth	enum-	
	varchar2(50)	PROD_GA50X4	N	1; 1.5; 2; 3; 4; 6; 8;	
	Other				
		Length (Feet):	drop-down length	enum-	
45	varchar2(50)	PROD_GA50X5	N	1; 2; 3; 4; 5; 6; 10;	
	Other				
	Electrical - Other 1327 1355 ELOTHERR	Type:	drop-down		
	type enum-varchar2(50)	PROD_GA50X1	N		Other

## A Sample User Interface Configuration File

One embodiment of the invention generates 700-710 an ontology.properties file which includes the content shown below; for conciseness, material similar to that shown was omitted at the points indicated by ellipses and some blank lines were removed. Similar examples are

5 provided in incorporated provisional application 60/280,196 at pages 31-33 and 61-63:

# The Category ID "0" is reserved for the Generic Attributes

CSEARCH0=Generic

CMDISP0=

10 CRDISP0=

TTL\_CATEGORIES=22

# Category names for search and display on the UI

15

CSEARCH1=Alkaline Batteries

CMDISP1=Alkaline Batteries

CRDISP1=Alkaline Batteries

CORDER=1

20

CSEARCH2=Desk Staplers

CMDISP2=Desk Staplers

CRDISP2=Desk Staplers

CORDER=2

25

...

CSEARCH22=Paper Clips

CMDISP22=Paper Clips

30

CRDISP22=Paper Clips

CORDER=22

# All the Attributes for the above Categories

# First the Generic attributes

35

VAR0A0=

DISP0A0=

ZONE0A0=

PIMS0A0=prod\_name

40

ISSEARCH0A0=false

FORMTYPE0A0=

VAR0A1=

DISP0A1=Description

45

ZONE0A1=

PIMS0A1=product\_description

ISSEARCH0A1=false

FORMTYPE0A1=

VAR0A2=  
DISP0A2=Supplier  
ZONE0A2=  
5 PIMS0A2=supplier\_short\_name  
ISSEARCH0A2=false  
FORMTYPE0A2=

VAR0A3=  
10 DISP0A3=Supplier Catalog #  
ZONE0A3=  
PIMS0A3=supplier\_cat\_num  
ISSEARCH0A3=false  
FORMTYPE0A3=

15 ...

VAR0A17=  
DISP0A17=  
20 ZONE0A17=  
PIMS0A17=supplier\_id  
ISSEARCH0A17=false  
FORMTYPE0A17=

25 TTL0=18

# Now the category specific attributes

#1 Alkaline Batteries

30 VAR1A0=size  
DISP1A0=Size  
ZONE1A0=size  
ISSEARCH1A0=true  
35 FORMTYPE1A0=enum

VAR1A1=mercuryFree  
DISP1A1=Mercury Free?  
ZONE1A1=mercury\_free  
40 ISSEARCH1A1=true  
FORMTYPE1A1=radio

TTL1=2

45 #2 Desk Staplers

VAR2A0=color  
DISP2A0=Color  
ZONE2A0=color  
50 ISSEARCH2A0=true  
FORMTYPE2A0=text

```

VAR2A1=stapleType
DISP2A1=Staple Type
ZONE2A1=staple_type
5 ISSEARCH2A1=true
FORMTYPE2A1=enum

VAR2A2=paperCapacity
DISP2A2=Paper Capacity
10 ZONE2A2=paper_capacity
ISSEARCH2A2=true
FORMTYPE2A2=text

TTL2=3
15 ...
#22 Paper Clips

20 VAR22A0=size
DISP22A0=Size
ZONE22A0=size
ISSEARCH22A0=true
FORMTYPE22A0=enum
25 VAR22A1=material
DISP22A1=Material
ZONE22A1=material
ISSEARCH22A1=true
30 FORMTYPE22A1=enum

VAR22A2=color
DISP22A2=Color/Finish
ZONE22A2=color
35 ISSEARCH22A2=true
FORMTYPE22A2=text

TTL22=3
40 20

```

### A Sample Catalog Enumeration Load Sheet

One embodiment of the invention generates 714 a generic attributes enum load file. The data in the load file is generated in the following format:

45 'PIMS'|'1023'|'PROD\_GA50X1'|'7'|'Neomycin';

where PIMS stands for the project name, 1013 stands for the Prod\_Att\_Def\_Id, PROD\_GA50X1 is the actual column name, 7 is an example of the enumeration value, it could be any numeric number, and the last entry is the actual value (human-oriented representation).

One enum table includes an entry set that can map from default\_node\_id to prod\_attr\_def\_id codes; an entry set that can map from default\_node\_id to visual representation; and an entry set that can map from prod\_attr\_def\_id to visual representation.

## 5 A Sample Checklist

Figures 9A and 9B of incorporated provisional application no. 60/274,595 show an example from a checklist generated 718 in spreadsheet form from an express structural ontology specification 406. For convenience, that example is also described here. The checklist file contains a single worksheet, called “checkList”. An initial part of the worksheet contains four named columns, including an empty “Comments” column and three columns containing values such as the following:

<u>Category Name</u>	<u>Spelling correct?</u>		<u>Format correct?</u>	
Respiratory Therapy	yes	no	yes	no
Plastics	yes	no	yes	no
Nutrition and Dietary Supplies	yes	no	yes	no
Veterinary Supplies and Equipment	yes	no	yes	no

Subsequent rows ask “Are the categories in the correct order of appearance? yes no”, and “Is the list of categories complete? yes no”. This is followed by “Category names are spelled correctly, are properly formatted and are in the correct order on the Category Search Page”.

A next portion, “QA of Category Search Pages by Family”, contains columns named “Category Name”, “Attribute Name”, “Correct?”, “Search type”, “Correct”, “Allow enum values”, “Correct?”, and “Comments”. Within these columns the categories and attributes of the marketplace (as extracted from the specification 406) are listed for review. For instance, a

25 Category Name “Respiratory Therapy” has associated rows with Attribute Name cell values such as “respiratory\_type” and “mdi\_field\_1”, with Search type cell values “drop-down” and “none”, and with an Allow enum values cell containing “Aerosol Therapy/Treatment Devices; Air Compressors and Pumps; Air Oxygen Mixers, Concentrators, Purifiers, Cleaners, and Analyzers; Airway Kits; Blood Gas Sampling Equipment and Supplies; Breathing Aids and

30 Exercisers; Filters, Moisture Traps; Gas Monitoring Equipment; Gases, Regulators, Cylinders and Accessories; Humidifiers; Iron Lungs; Other; Oxygen Masks and Delivery Devices; Percussors; Pulmonary Function Testing Devices--Peak Flowmeters, Volumeters, Spirometers; Respirators and Ventilators and Accessories; Vaporizers”. Cells at the end of the Respiratory Therapy section ask “Attribute list complete? yes no” and “Enum lists complete yes no”, so the person inspecting the attributes and enumeration values can circle “yes” or

“no” accordingly. Similar sections are provided for other Category Names. Other examples are provided in incorporated provisional application 60/280,196 at pages 26-28 and 67-68.

### **Additional Examples and Details**

5 File naming conventions may be used. For instance, the PROD\_ATT\_DEF output files created by an inventive script 414 may be named according to the format <three-character alphabetic vertical code>\_load\_setup\_<NN>.txt. Various delimiter characters may be used between parameters in the script command line, with a blank space preferred. One file naming convention for output files created by inventive scripts 414 has the vertical company name,  
10 then an underbar, then the type of printfile, then another underbar, and finally the “.out” filename extension.

In one embodiment, temporary files used by a script according to the invention are placed in a directory; files are organized according to families. For instance, a generated 712, 716 Acmotor.txt file contains the following excerpt:

15 type|char|50||Squirrel Cage Induction Motor;Synchronous Motor;Other|prod\_ga50x1  
speeds|char|50||Single Speed;Two Speed;Other|prod\_ga50x2  
enclosure|char|50||TEFC;Explosion Proof (FCXP);Drip Proof (DP);Open;Other|prod\_ga50x3  
voltage|char|50||460 Volts;115/230V;190/380V;200V;200/400V;230/460V;208-  
20 230/460;Other|prod\_ga50x4  
phase|char|50||Single Phase, Capacitor Start;Single Phase, Split Phase;Three  
Phase;Other|prod\_ga50x5  
frequency|char|50||60;50;Other|prod\_ga50x6  
service\_factor|char|50||1.15;1.0;Other|prod\_ga50x7  
25 hp|char|50||0.5;0.75;1;1.5;2;3;5;7.5;10;15;20;25;30;40;50;60;75;100;125;150;200;250;Other|pr  
od\_ga50x8  
rpm|char|50||3600;1800;1200;900;Other|prod\_ga50x9

In general, the invention may be implemented in various ways using various data formats. For instance, one implementation might store generated 712, 716 data such as that just illustrated for Acmotor.txt in a spreadsheet file (e.g., Acmotor.xls) rather than (or in addition to) storing that data in a text file.

A script 414 may be used to create other scripts 414, 808. For instance, in one embodiment scripts are created 726 by giving the following command:

35 auto.ksh “family name” “default node id” “product att def id”

where an example of a “family name” is “bolts”, an example of a “default node id” is “1031”, and an example of a “product att def id” is “1456”.

One previously internal document describing a vertical generation script 414 according to the invention for use on a UNIX system, such as a UNIX Solaris system, includes the text

shown below [reference numbers have been added for convenience]; a shorter version of this document is also recited in incorporated provisional application 60/278,558 at pages 5-9:

### How do you use the scripts

- 5    The main menu, menu\_vertical.ksh, calls the vertical generating perl scripts which are in /CES/users/sma/verticals\_build directory. You can cvs check out the scripts from directory ces/ventro/scripts/pims\_1\_7, and point BuildOut link to the directory you checked out.
- 10    1. Following steps show you how to create an alias in your .cshrc file for easy access the menu\_vertical.ksh script.
- a). vi .cshrc  
      b). add the line “alias BuildOut /CES/users/sma/verticals\_build/menu\_vertical.ksh”  
      c). or instead of doing step 2, you can create a BuildOut alias to the directory where the perl scripts are cvs checked out to.  
      d). source .cshrc (now BuildOut will associate with the path of perl scripts, you can type in the alias and the script will be called correctly.  
      e). you can now start your vertical BuildOut script.
- 15

### Save the ontology file as a text file

- 20    1. create a new directory using Unix command- mkdir \$verticalName, \$verticalName can be marketmile,promedix and etc  
      2. change permission of the directory to 775 (chmod 775 \$verticalName)  
      3. rename the ontology file as tab-delimited file with the extension .txt and save it under the \$verticalName directory.  
      4. modify the tab-delimited file and delete all the headings from the files. (because each vertical may add different headings in the ontology file, the perl script can not catch all the exception). Save the modified file.  
      e). You can run the BuildOut from any directory you want

30    How to run BuildOut script

- a) If the ontology file named “\$vertical.txt” is under the directory TestVerticals, you can use this ontology file as the source file to build out all the data load files and perl scripts for the \$vertical.  
      b) There are 2 types of choices. You can build out [506] individual files or scripts by choose one of the number from the Menu lists. Or, you can choose number 14 which builds out [506] the entire vertical suites.  
      c) You need to give the absolute directory path where the tab-delimited ontology file located, and give the file name as well. (Remember to remove the headings from the tab-delimited file before running the BuildOut)  
      d) After finishing the BuildOut script, a report file in the same directory as ontology file is generated which briefly explains you what the files are.

### Run BuildOut

45    ~ /TestVerticals > **BuildOut**

## Menu lists

1. Convert ontology txt file to flat txt file
2. Generate enum data file from ontology spec sheet [406]
- 5 3. Generate [714] qa tags from ontology spec sheet [406]
4. Generate UI validation file from ontology spec sheet [406]
5. Generate [716] data files for prod\_tree\_def and prod\_att\_def tables
- 10 6. Generate printfile modules from ontology spec sheet
7. Generate [726] parsing template from ontology spec sheet
8. Generate Excel spread sheets from ontology spec sheet
9. Generate create\_new\_company.ksh
- 15 10. Generate [726] tab2pipeTemplate.pl
14. Create verticals

Please choose one of the above number [-1]:

20 14

Please enter absolute directory path of the ontology file  
: /CES/users/sma/TestVerticals  
Enter the name of the ontology text file []: mmi.txt

25 Directory ..... /CES/users/sma/TestVerticals  
Spec sheet file name ..... mmi.txt  
Output file name .....  
Option ..... Build up [506] all enum files, qa  
tags, UI validation file, data input files for  
30 prod\_tree\_def, data file for prod\_att\_def tables, print  
modules, standard parsing script, excel files for DBA  
shares

Are they correct [y]? y

35 sma@indy:~/TestVerticals > ls -ltr  
total 296  
-rwxr-xr-x 1 sma staff 13072 Oct 9 11:39  
mmi.txt\*  
40 drwxr-xr-x 2 sma staff 4096 Oct 11 17:31  
all\_tags/  
-rw-r--r-- 1 sma staff 5739 Oct 11 17:37  
enumFile\_for\_review.out  
-rw-r--r-- 1 sma staff 3299 Oct 11 17:37  
45 enumFile [712, 728]  
-rw-r--r-- 1 sma staff 1739 Oct 11 17:37  
resetArray\_Verical.pm [726]  
-rwxr-xr-x 1 sma staff 11470 Oct 11 17:37  
qaTemplate.pl\* [724]  
50 -rw-r--r-- 1 sma staff 562 Oct 11 17:37  
prod\_tree\_def.out [712 and/or 728]

	-rw-r--r--	1	sma	staff	37112	Oct 11 17:37
	prod_att_def.out	[712 and/or 728]				
5	-rw-r--r--	1	sma	staff	4930	Oct 11 17:37
	print4PIMS_vertical.pm	[726]				
	-rw-r--r--	1	sma	staff	8719	Oct 11 17:37
	checkList.xls	[718]				
	-rw-r--r--	1	sma	staff	6050	Oct 11 17:37
	Standard_paringTemplate.pl	[726]				
10	-rwxr-xr-x	1	sma	staff	619	Oct 11 17:37
	tab2pipeTemplate.pl*	[726]				
	-rw-r--r--	1	sma	staff	1066	Oct 11 17:37
	report.txt					
	-rwxr-xr-x	1	sma	staff	1175	Oct 11 17:37
15	new_companyTemplate.ksh*	[726]				
	drwxr-xr-x	2	sma	staff	4096	Oct 11 17:37
	family_Attribute/					

### Detail about the scripts and data files.

- 20
- **Script: qaTemplate.pl**
  - **Purpose:** The data files are pipe-delimited text files which are ready to load into databases by load scripts. Since the nature of data files is complicated, we like to ensure the data schema in the text files are correct before being loaded to production databases. The [generated 724] qa script, qaTemplate.pl, is designed to validate individual columns of data files to make sure the data types and values of the data load files are the same as defined by ontology. Data load files contain 2 major portions, the generic attributes and the family attributes. There are 44 generic attributes for PIMS 1.7 schema, and the numbers of family attributes are vertical dependent. The number of generic attributes can be updated according to the different versions of PIMS.
  - **QaTemplate.pl reads tag files under all\_tags directory:** Before building qaTemplate.pl, the tag\_generator.pl gets called first to build all the tag files based on ontology file and stored them under the all\_tags directory. The tag files have names with \$printfile.txt which will be used for matching data files during qa.
  - **How to modify the qaTemplate.pl script:**
- 40           Before running qaTemplate.pl, you need to modify the following items.

#### Note 1:

45           (Required) you need to update the \$qaDir to where the qaTemplate.pl script is created.

```
my $qaDir = "/CES/users/<changeme>";
```

e.g. If the qaTemplate.pl was created under /CES/users/sma/marketmile/verticals, then the \$qaDir will be

50           \$qaDir = "/CES/users/sma/marketmile/verticals";

**Note 2:**

5

(Optional) I suggest to rename qaTemplate.pl as qa\$Vertical.pl, it will help you identify different tag sources for qaTemplate.pl.

e.g. mv qaTemplate.pl qaMmi.pl (for \$Vertical name as Mmi)

**Note 3:**

10

(Optional) Create an alias in your login .cshrc file which allows easy access of qa script.

15

alias qaMmi.pl /CES/users/sma/marketmile/verticals/qaMMI.pl

**Note 4:**

20

(Optional) You can cd to the directory where the data files are, and run the following command.

Usages: qaMmi.pl <company name> /\*out  
|←---- \$ARGV[0] --→|

**Note 5:**

25

The qaTemplate.pl reads and extracts the \$printfile names from all the data load files provided by \$ARGV[0]. It is important to have \$printfile in the tag files match \$printfile in the data load files. If the spelling of data files don't have correct \$printfile as provided in ontology, qaTemplate.pl will fail to qa the data files.

30

35

After matching the data load files with one of the tag files, qaTemplate.pl compares both generic and family attributes in the data files with both generic.txt and the chosen tag file. The generic.txt is common across all the verticals which describes the schema of data types, allow values and required-or-not of all the generic attributes. Family tag files contains the data schema of the family attributes only.

e.g. The qaTemplate.pl extracts \$printfile names from data files

```
if ($infile =~ /\Q$companyName\E\_(.+)\.out/) {
    $family = $1;
    // $1 contains the $printfile which is shortname for the $family
    .....
}
```

40

45

e.g. The qaTemplate.pl extracts \$printfile names from tag files to build hash %wantedTagHash

50

```
foreach $tagFileName (@tagFiles) {
```

```
open (TAG, $tagFileName) || die "Can't open $tagFileName $! \n";
$tagFileName =~ s/.+V(.*)\.\txt/$1/i;
```

.....

```
5           $wantedTagsHash{$tagFileName} {"columnCt"} = $line;
}
```

- **Script:** **tab2pipe.pl**

10 **Usages:** tab2pipe.pl <source directory> <new directory>

**Purpose:** [Generated 726] tab2pipe.pl is to convert tab-delimited files with pipe delimiter, and the file can have extension txt, out, inp and ctl.

15 **Note 1:** The script will convert the tab to pipe, and remove ctl M to replace it with a new line for all the files under the “source directory” and copy the edited files to a new directory.

**Note 2:** You need to provide the absolute paths of the source and new directories.

20 **Note 3:** The new directory will be created if it is not there previously.

- **Script:** **Standard\_paringTemplate.pl**

- **Purpose:** This is a perl parsing script which calls print modules. It takes 6 steps to generate [726] the standard\_parsingTemplate.pl

25 Step 1: Saved the ontology file as text file.

Step 2: merge.pl will merge the text file as a new input file.

30 Step 3: build\_printColumnCtHash.pl builds a “printfile hash” with \$printfile as key, and list reference as values. The list contains total attribute counts, prod\_att\_def\_id, and default\_node\_id.

Step 4: combineResetArray joins pimsColumnCt\_table with resetArray.pm to form [726] vertical specific resetArray.pm.

Step 5: parsingTemplate\_1.pl and parsingTemplate\_2.pl contain portions of parsingTemplate.

35 Step 6: combineStdParsing.ksh joins both parsingTemplate together.

**Note 1:** This is a parsing template. The script calls the print module and you need to update all the <changeme>.

40 • **Script:** **tag\_generatorTemplate.pl**

- **Directory:** all\_tags

Note 1: The directory contains all the data files used by the qa scripts. It was generated by tag\_generatorTemplate.pl

45

5 Note 2: tag\_generatorTemplate.pl reads in ontology flat file to build qa files from ontology spreadsheet. All the qa files are stored under “all\_tags” directory, and the qa files are named as \$printfile.txt such as pipe.txt. The format of the \$printfile.txt is “\$famAtt \$dataType \$dbSize \$require \$allowValue \$pimsType &&&&”. The qaTemplate.pl reads in the qa files and parses the values with pipe-delimiter. After comparing the data load files again their corresponding qa files, the script will report errors in genericError file.

- **Script:** mapExcelfile.pl

10 Purpose: to build [712] family\_attribute excel spreadsheets and node.ga files from ontology spreadsheet.

15 Note 1: The mapExcelfile.pl reads ontology data and generic.txt to build 2 hash tables including the generic hash and family hash. Generic hash contains all the common generic attributes, and family hash contains the family attributes described in ontology spec sheet.

- **Directory:** family\_Attribute

20 Note 1: The directory contains all the Excel spread sheets used by Database engineering team to create vertical specific schema.

- **Directory:** node\_ga

25 Note1: [generated 712, 714] node\_id.ga file, used by DBA team to create [726] data load files

- **File:** report.txt

30 Note 1: The file contains the paths of the new files generated by menu\_vertical.ksh

- **Script:** validate.pl

Purpose: validate.pl reads family\_attributes, search\_type and allowed\_values from ontology spec sheet to build [718-722] a checkList in Excel format. The checkList.xls is used for front end UI display.

35

- **File:** checkList.xls

Note 1: The file contains the checklist of the [generated 720-722] front end values.

40 • **Files:** enumFile and enumFile\_for\_review.txt

Note 1: enumFile is [generated 712, 728] for DBA load to enum table

Note 2: enumFile\_for\_review.txt is for content QA do the data review

45 • **File:** prod\_att\_def.out

Note 1: [Generated 712, 728] For DBA to populate prod\_att\_def table

- **File:** prod\_tree\_def.out

50

Note1: [Generated 712, 728] For DBA to populate prod\_tree\_def table.

- **File:** ontology.properties

5 Note1: For front end to use. It is B20 compatible.

- **File:** Enum.properties

Note1: For front end enum get service. It is B20 compatible.

- **Script:** resetArrayTemplate.pm

- **Purpose:** To reset the generic and family arrays to values null.

- **Note 1:** combineResetArray.ksh joins lines 1-36 of resetArrayTemplate.pm, with

- **Note 2:** script build\_printColumnCtHash.pl creates “printfile” as a key and 3 values as hash values. The values are total column counts, prod\_att\_def\_id, and default\_node\_id.

```
"ADAPTER" => [46,1026,1054],  
"BEND" => [45,1027,1055],  
"BOLT" => [45,1023,1051],
```

- **Script:** familyAtt\_Template.pl

Purpose: familyAtt\_Template.pl reads prod\_att\_def\_id and default\_node\_id columns from ontology spec sheet, and generates two files including “prod\_tree\_def.out” and “load\_setup.out”. Prod\_tree\_def.out is used by DBE to populate prod\_tree\_def table, and the data is required for database build out. Load\_setup.out is used by DBE to create [726] automated data load scripts.

30 In one embodiment, a menu\_vertical.ksh file contains text such as the following:

```
#!/usr/bin/sh -f  
  
*****  
#  
#  
35 # Name:  
#      vertical.sh  
#  
# Purpose:  
#  
40 # Usage:  
#      Run this script by typing vertical.sh  
#  
*****  
  
45 unalias cd  
unalias rm  
unalias cp  
  
bold=`tput smso`
```

```
offbold=`tput rmso`  
confirm="n"  
  
5 PROJECT_DIR="/CES/users/sma/vertical_buildPIMS17";  
  
while [ "$confirm" != "" ] && [ "$confirm" != "y" ]  
do  
  
10 clear  
  
echo  
"+=====  
echo "|  
15 echo "| Vertical menus  
echo "|  
echo "|  
echo "+=====  
  
20 echo " "  
echo " "  
  
echo " "  
echo " "  
  
25 input=""  
while [ "$input" = "" ]  
do  
    OPTION_FLAG=-1; export OPTION_FLAG  
  
30     clear  
     echo " "  
     echo " "  
     echo " " Menu lists"  
35     echo " "  
     echo " "  
  
     echo "1. Convert ontology txt file to flat txt file"  
     echo ""  
40     echo "2. Generate enum data file from ontology spec  
sheet"  
     echo ""  
     echo "3. Generate qa tags from ontology spec sheet"  
     echo ""  
45     echo "4. Generate UI validation file from ontology spec  
sheet"  
     echo " "  
     echo "5. Generate [726] data files for prod_tree_def  
and prod_att_def tables, and load_setup.out for auto.ksh"  
50     echo " "
```

```
        echo "6. Generate printfile modules from ontology spec
sheet"
        echo ""
        echo "7. Generate parsing template from ontology spec
5 sheet"
        echo ""
        echo "8. Generate Excel spread sheets/node_id.ga files
from ontology spec sheet"
        echo ""
10       echo "9. Generate [726] create_new_company.ksh"
        echo ""
        echo "10. Generate tab2pipeTemplate.pl"
        echo ""
        echo "11. Generate ontology.properties"
15       echo ""
        echo "14. Create verticals"
        echo ""

        echo "Please choose one of the above number
20      [$OPTION_FLAG]:\c"
        echo ""

        read input
        if [ "$input" = "" ]; then
25          input=$OPTION_FLAG
        fi

        if [ "$input" <> "" ]; then

30          if [ "$input" = 1 ]; then
              ACTION="Convert ontology spec sheet to flat file"
              OPTION="1"

            elif [ "$input" = 2 ]; then
35              ACTION="Generate enum file from ontology spec
sheet"
              OPTION="2"
            elif [ "$input" = 3 ]; then
              ACTION="Generate qa tags from ontology spec sheet"
              OPTION="3"

40            elif [ "$input" = 4 ]; then
              ACTION="Generate qa tags from ontology spec sheet"
              OPTION="4"

45            elif [ "$input" = 5 ]; then
              ACTION="Generate prod_att_def and pord_tree_def
files"
              OPTION="5"
            elif [ "$input" = 6 ]; then
```

```
        ACTION="Generate print modules from ontology spec
sheet"
        OPTION="6"

5          elif [ "$input" = 7 ]; then
            ACTION="Generate a parsing template from ontology
spec sheet"
            OPTION="7"

10         elif [ "$input" = 8 ]; then
            ACTION="Generate Excel templates/node_id.ga from
ontology spec sheet"
            OPTION="8"

15         elif [ "$input" = 9 ]; then
            ACTION="Generate the create_new_company.ksh"
            OPTION="9"

20         elif [ "$input" = 10 ]; then
            ACTION="Generate the tab2pipe.pl"
            OPTION="10"

25         elif [ "$input" = 11 ]; then
            ACTION="Generate the ontology.properties"
            OPTION="11"

30         elif [ "$input" = 14 ]; then
            ACTION="Build up all enum files, qa tags, UI
validation file, data input files for prod_tree_def, data file
for prod_att_def tables, print modules, standard parsing
script, excel files for DBA shares"
            OPTION="14"

35             fi
             fi

done

40     input=""
     while [ "$input" = "" ]
     do
        echo "Please enter absolute directory path of the ontology
file : \c"
        read input
45
        if [ "$input" <> "" ]; then
            if [ -d $input ] ; then
                TOP=$input; export TOP
                break
50
            else
                echo "'$input' is not a valid directory."
```

```
        input=""
        echo " "
        fi
    fi
5   done

    input=""

10  if [ $OPTION != 9 ]; then
    while [ "$input" = "" ]
    do
        echo "Enter the name of the ontology text file
[$FILE_NAME]: \c"
15    read input
        if [ "$input" = "" ]; then
            input=$FILE_NAME
        fi

20    if [ "$input" <> "" ]; then
        if [ -r $TOP/$input ]
        then
            FILE_NAME=$input; export FILE_NAME
            break
        else
            echo "'$input' is not a valid file or file is
not readable."
            echo "All data files must under TOP/data
directory."
25    input=""
        echo " "
        fi
    done
30
35    fi

    if [ "$OPTION" != "3" ] && [ "$OPTION" != "5" ] && [
"$OPTION" != "7" ] && [ "$OPTION" != "8" ] && [ "$OPTION" !=
40 "7" ] && [ "$OPTION" != "6" ] && [ "$OPTION" != "14" ] && [
"$OPTION" != "9" ] && [ "$OPTION" != "10" ] && [ "$OPTION" !=
"11" ]; then

        input=""
45    while [ "$input" = "" ]
    do

        echo "Enter the output file name [$FLAT_FILE]: \c"
        read input
50    if [ "$input" = "" ]
        then
```

```

        FLAT_FILE="flat_spec.txt"; export FLAT_FILE
        break
        fi

5       if [ "$input" <> "" ]
then
        echo "HI IN THE LINE 214 $input $FLAT_FILE"
        FLAT_FILE=$input
        fi
10      done

        fi

        echo " "
15      echo " "

        echo " "
        echo " "
        echo "Directory ....." $STOP
20      echo "Spec sheet file name ....." $FILE_NAME
        if [ "$OPTION" != "3" ] && [ "$OPTION" != "5" ] && [
"$OPTION" != "6" ] && [ "$OPTION" != "7" ] && [ "$OPTION" !=
"9" ]
        then
25        echo "Output file name ....." $FLAT_FILE
        fi
        echo "Option ....." $ACTION
        echo ""

30        echo "Are they correct [y]?\\c"
        read confirm
        echo $confirm

        done
35        echo " "
        echo " "

        if [ "$OPTION" = "1" ]; then
            $PROJECT_DIR/merge.pl $STOP $FILE_NAME $FLAT_FILE
40
        elif [ "$OPTION" = "2" ]; then

            tempFile="tempflatfile.txt"
            $PROJECT_DIR/merge.pl $STOP $FILE_NAME $tempFile
45
            $PROJECT_DIR/enumTemplate.pl $STOP $tempFile $FLAT_FILE
            temp="$FLAT_FILE_for_review"
            echo "$temp"

50        echo "File $bold $FLAT_FILE $offbold is enum load file
which is in the $STOP directory"

```

```

        echo " "
        echo "File $bold $temp $offbold is for review purpose"
        echo " "

5   elif [ "$OPTION" = "3" ]; then
        tempFile="tempflatfile.txt"
        $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile
        $PROJECT_DIR/tag_generatorTemplate.pl $TOP $tempFile
        cp $PROJECT_DIR/qaTemplate.pl $TOP
10
        echo "Outputs are in $bold $TOP/all_tags directory
$offbold "
        echo "$bold qaTemplate is also copied to $TOP directory
$offbold"
15
        echo " "

elif [ "$OPTION" = "4" ]; then

        tempFile="tempflatfile.txt"
20
        $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile

        $PROJECT_DIR/validate.pl $TOP $tempFile $FLAT_FILE
        echo ""
        echo "UI validation file, $bold $TOP $FLAT_FILE.xls
$offbold "
25
        echo " "
        echo " "

elif [ "$OPTION" = "5" ]; then
30
        tempFile="tempflatfile.txt"
        $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile

        $PROJECT_DIR/familyAtt_Template.pl $TOP $tempFile
        echo ""
        echo "$bold $TOP/ prod_tree_def.out $offbold is generated
"
        echo "$bold $TOP/prod_att_def.out $offbold is generated
"
40
        echo " "
        echo " "

elif [ "$OPTION" = "6" ]; then
        ## create the hash for family
45
        tempFile="tempflatfile.txt"
        $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile
        $PROJECT_DIR/build_printColumnCtHash.pl $TOP $tempFile

        ## join the printColumnHash with the printTemplate
50
        $PROJECT_DIR/combinePrint4Pims.ksh $TOP pimsColumnCt_table

```

```

        echo "$bold print4PIMS_vertical.pm is created $offbold
under $TOP"

    elif [ "$OPTION" = "7" ]; then
5      ## hash for each family has to be built out first, so we
      then know
          ## how many column counts per family

          tempFile="tempflatfile.txt"
10     $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile
          $PROJECT_DIR/build_printColumnCtHash.pl $TOP $tempFile

          ## create the resetArray_Vertical.pm in the same directory
          ## join the hash value with the resetTemplate
15     $PROJECT_DIR/combineResetArray.ksh $TOP pimsColumnCt_table

          ## create the standard parsing template
          $PROJECT_DIR/parsingTemplate_1.pl $TOP
20     $PROJECT_DIR/parsingTemplate_2.pl $TOP $tempFile
          $PROJECT_DIR/combineStdParsing.ksh $TOP

    elif [ "$OPTION" = "8" ]; then
        tempFile="tempflatfile.txt"
25     $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile
        $PROJECT_DIR/mapExcelfile.pl $TOP $tempFile
        echo "excel files are created under $bold
$TOP/family_Attribue $offbold directory "
        echo "<NODE_ID>.ga files are created under $bold
30   $TOP/node_id $offbold directory "

    elif [ "$OPTION" = "9" ]; then
        echo "Create create_new_company.ksh"
        cp $PROJECT_DIR/new_companyTemplate.ksh $TOP
35

    elif [ "$OPTION" = "10" ]; then
        echo "Create tab2pipeTemplate.pl"
        cp $PROJECT_DIR/tab2pipeTemplate.pl $TOP

40   elif [ "$OPTION" = "11" ]; then
        tempFile="tempflatfile.txt"
        $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile
        $PROJECT_DIR/frontEndProperty.pl $TOP $FILE_NAME
        $PROJECT_DIR/combineFrontEnd.ksh $TOP
45   FE_property_family.txt FE_property_attribute.txt
        echo "excel ontology.properties file is created under $bold
$TOP/ontology.properties $offbold directory "

    elif [ "$OPTION" = "14" ]; then
50      tempFile="tempflatfile.txt"
      $PROJECT_DIR/merge.pl $TOP $FILE_NAME $tempFile

```

```

## create enum file
$PROJECT_DIR/enumTemplate.pl $TOP $tempFile $FLAT_FILE
5
## create qa scripts
$PROJECT_DIR/tag_generatorTemplate.pl $TOP $tempFile
cp $PROJECT_DIR/qaTemplate.pl $TOP

10 ## create validation file
checkList="checkList"
$PROJECT_DIR/validate.pl $TOP $tempFile $checkList

## create files for prod_tree_def and prod_att_def tables
15 $PROJECT_DIR/familyAtt_Template.pl $TOP $tempFile

## create print modules
$PROJECT_DIR/build_printColumnCtHash.pl $TOP $tempFile
$PROJECT_DIR/combinePrint4Pims.ksh $TOP pimsColumnCt_table
20

## create standard parsing script

$PROJECT_DIR/combineResetArray.ksh $TOP pimsColumnCt_table
$PROJECT_DIR/parsingTemplate_1.pl $TOP
25 $PROJECT_DIR/parsingTemplate_2.pl $TOP $tempFile
$PROJECT_DIR/combineStdParsing.ksh $TOP

## create Excel spreadsheets
$PROJECT_DIR/mapExcelfile.pl $TOP $tempFile
30

## copy tab2pipeTemplate.pl and new_companyTemplate.ksh

cp $PROJECT_DIR/new_companyTemplate.ksh $TOP
cp $PROJECT_DIR/tab2pipeTemplate.pl $TOP
35

## create front end ontology.properties

$PROJECT_DIR/frontEndProperty.pl $TOP $tempFile
$PROJECT_DIR/combineFrontEnd.ksh $TOP
40 FE_property_family.txt FE_property_attribute.txt

## clean up the temp files
`rm $TOP/temp1_std_parsing.txt`
`rm $TOP/temp2_std_parsing.txt`
`rm $TOP/tempflatfile.txt`
`rm $TOP/pimsColumnCt_table`

45

## generate a report

50 echo " " > $TOP/report.txt
echo " " >> $TOP/report.txt

```

```
      echo "           Reports for the created files (PIMS 1.7)"  
    >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
5       echo " " >> $TOP/report.txt  
  
      echo "1. enumFile_for_review.out --> is the enum file for  
qa review\n" >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
10      echo " " >> $TOP/report.txt  
      echo "2. enumFile --> is the enum file for loading to  
database\n\n" >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
15      echo "3. prod_tree_def.out --> is the data file for  
loading to prod_tree_def table\n\n" >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
      echo "4. prod_att_def.out --> is the data file for  
loading to prod_att_def table\n\n" >> $TOP/report.txt  
20      echo " " >> $TOP/report.txt  
      echo "5a. Directory $TOP/all_tags contains all the  
configuration files for QA purpose\n\n" >> $TOP/report.txt  
      echo "5b. [generated 724] qaTemplate --> is the qa script.  
please remember to update the $qaDir in the qaTemplate\n\n" >>  
25 $TOP/report.txt  
      echo " " >> $TOP/report.txt  
      echo "6. UI validation file --> is a file used for front  
end validation" >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
30      echo "7a. Standard_paringTemplate.pl --> the perl script  
is the Standard parsing template. Please remember to modify  
the script\n" >> $TOP/report.txt  
      echo "7b. There are 2 [generated 726] modules for the  
Standard_parsingTemplate.pl. --> resetArray_Vertical.pm and  
35 print4PIMS_vertical.pm\n\n" >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
      echo "8. Directory $TOP/family_Attribute contains all the  
Excel spreadsheets/NODE_GA for the ontology " >>  
$TOP/report.txt  
40      echo "8. Directory $TOP/node_ga contains all the node.ga  
files for database loading " >> $TOP/report.txt  
      echo " " >> $TOP/report.txt  
      echo "9. Generate a shell script, new_companyTemplate.ksh"  
>> $TOP/report.txt >> $TOP/report.txt  
45      echo " " >> $TOP/report.txt  
      echo "10. Generate a perl script, tab2pipeTemplate.pl " >>  
$TOP/report.txt  
      echo " " >> $TOP/report.txt  
  
50      echo "11a. Generate B20 ontology.properties which is used  
for front end UI display " >> $TOP/report.txt
```

```

        echo "11b. Generate B20 Enum.properties which is used for
EnumGet service " >> $TOP/report.txt
        echo " " >> $TOP/report.txt
5
        echo "Vertical files have been created, please read
$TOP/report.txt to get detail information"
        fi
10
exit 0

```

In one embodiment, a generated 726 load\_setup.out file contains text such as the  
15 following excerpt:

```

auto.ksh pbball 1085 1057
auto.ksh pbfly 1273 1246
auto.ksh pvcheck 1274 1247
auto.ksh pvdiph 1275 1248
20 auto.ksh pvgate 1276 1249
auto.ksh pvglobe 1277 1250
auto.ksh pvneedle 1278 1251

```

A database group may use files “load\_setup.out” and node\_ga files \*.ga and  
25 automatically create 726 database load scripts from the configuration files based on the  
ontology specification 406. In one embodiment, the auto.ksh file includes the following:

```

#!/bin/ksh
ORACLE_HOME=/opt/oracle/815
TNS_ADMIN=/var/opt/oracle; export TNS_ADMIN
30 PATH=/home/bpatel/db_eng/pims17/load/common/template:$ORACLE_HOME/bin:/bin:/usr/bin:/etc; export PATH
LD_LIBRARY_PATH=$ORACLE_HOME/lib; export LD_LIBRARY_PATH

if [ $# -ne 7 ]
35 then
    echo
        echo "USAGE: $0 <family><node_id> <att_def_id><db_name>
<3 digit vertical code> <pims_load a/c password> <pims a/c
password>"
40        echo "FAMILY      --> Name of the family ex. alk_bat"
        echo "NODE_ID      --> Node ID as specified in
prod_tree_def ex. 1120"
        echo "ATT_DEF_ID --> Attribute definition id ex. 1101"
        echo "DB_NAME      --> Name of the database in
45 which data will be loaded"

```

```

        echo "VERT      --> Three digit Vertical code ex. cmd
for chemdex"
        echo "PASSWORD_LOAD    --> Password for the PIMS_LOAD
account"
5       echo "PASSWORD_PIMS    --> Password for the PIMS
account"
        echo
        exit -1
else
10      FAMILY=`echo $1|cut -c1-25`
        NODE_ID=$2
        ATT_DEF_ID=$3
        DB_NAME=$4
        VERT=$5
15      PASSWORD_LOAD=$6
        PASSWORD_PIMS=$7
fi
COMMON_BASE=/home/bpatel/db_eng/pims17/load/common/template
VERT_BASE=/home/bpatel/vert_loads/pims17/$VERT
20      cd $VERT_BASE/tmp

# Creating the GA attributes line here
sed s/" "//g $VERT_BASE/ga/$NODE_ID.ga >att_def02
awk -f $COMMON_BASE/combine.awk att_def02 >att_def03
25      awk -f $COMMON_BASE/combine_val.awk att_def02 >att_def_val03
rm att_def02

# Creating GA attribute description here
sqlplus -s pims/$PASSWORD_PIMS@$DB_NAME
30      @$COMMON_BASE/get_ga_desc.sql $ATT_DEF_ID
        sed /SQL/d att_desc.lst >att_desc01
        sed /:/d att_desc.lst >att_desc01
        sed s/" "//g att_desc01 >att_desc02
        awk -f $COMMON_BASE/combine.awk att_desc02 >att_desc03
35      sed s/NOTNULL/"NOT NULL"/g att_desc03 >att_desc04
        rm att_desc.lst att_desc01 att_desc02 att_desc03

# Creating the loader file here
sed s/ADD_TEMPLATE_HERE/$FAMILY/g $COMMON_BASE/load_template.sh
40      >../loader/load_$FAMILY.sh
        chmod 755 ..../loader/load_$FAMILY.sh

# Creating the loader control file here
sed s/ADD_TEMPLATE_HERE/$FAMILY/g $COMMON_BASE/template.ctl
45      >$FAMILY.ctl
        GA_LINE=`cat att_def03`
        sed s/ADD_GA_HERE/$GA_LINE/g $FAMILY.ctl >../ctl/$FAMILY.ctl
        rm $FAMILY.ctl

50      # Creating Table file here
        GA_LINE=`cat att_desc04`
```

```

sed s/ADD_TEMPLATE_HERE/$FAMILY/g $COMMON_BASE/template.sql
>tab01.sql
sed s/ADD_GA_HERE/"$GA_LINE"/g tab01.sql >../table/$FAMILY.sql
sqlplus -s pims_load/$PASSWORD_LOAD@$DB_NAME
5 @$VERT_BASE/table/$FAMILY.sql
rm tab01.sql

# Creating mapping function here
GA_DEF=`cat att_def03`
10 GA_VAL=`cat att_def_val03`
sed s/ADD_TEMPLATE_HERE/$FAMILY/g $COMMON_BASE/map_template.sql
>map01.sql
sed s/ADD_ATT_VAL_HERE/$ATT_DEF_ID/g map01.sql >map02.sql
sed s/ADD_NODE_VAL_HERE/$NODE_ID/g map02.sql >map03.sql
15 sed s/ADD_PROD_GA_HERE/"$GA_DEF"/g map03.sql >map04.sql
sed s/ADD_PROD_GA_VAL_HERE/"$GA_VAL"/g map04.sql
>../map/map_$FAMILY.sql
sqlplus -s pims_load/$PASSWORD_LOAD@$DB_NAME
@$VERT_BASE/map/map_$FAMILY.sql
20 rm map01.sql map02.sql map03.sql map04.sql

# Done creating all files, Removing the junk
rm att_def03 att_def_val03 att_desc04

```

25 In one embodiment, a generated 712, 714 .ga file contains the following:

```

PROD_GA50X1
PROD_GA50X2
PROD_GA50X3
PROD_GA50X4
30 PROD_GA50X5
PROD_GA50X6
PROD_GA1X7
PROD_GA1X8
PROD_GA1X9
35 PROD_GA50X7
PROD_GA1X10

```

In one embodiment, the load-setup.out is a shell script file generated 506 using the vertical buildout scripts, which the DBA runs to create the data loading scripts used by the 40 marketplace. This script uses the .ga files and a DBA-created script called auto.ksh to automatically generate 506 the ontology-specific data load scripts for a marketplace. It is a script that uses config files and a generic set of scripts to create customized scripts for data loading.

45 In one embodiment, each line of the load\_setup.out shell script builds a different family-specific data load script.

One embodiment includes one .ga file per product family.

In one embodiment, a combine.awk file referenced by auto.ksh includes:

```
{cmd=cmd ", " $0}
END {print cmd}
```

5 In one embodiment, a combine\_val.awk file referenced by auto.ksh includes:

```
{cmd=cmd ", l_record." $0}
END {print cmd}
```

In one embodiment, a get\_ga\_desc.sql file referenced by auto.ksh includes:

```
10 SET NEWPAGE 0
SET TAB OFF
SET SPACE 0
SET LINESIZE 132
15 SET PAGESIZE 0
SET ECHO OFF
SET FEEDBACK OFF
SET HEADING OFF
spool att_desc
20 select COL_NAME||' varchar2('||to_char(data_size)||')
      '||decode(required,'Y','NOT NULL','NULL') from
prod_att_def
where prod_att_def_id=&1
and COL_NAME not in (
25 'PROD_GA500X1',
'PROD_GA500X2',
'PROD_GA500X3',
'PROD_GA1X1',
'PROD_GA1X2',
30 'PROD_GA1X3',
'PROD_GA500X4',
'PROD_GA250X1',
'PROD_GA1X4',
'PROD_GA250X2',
35 'PROD_GA1X5',
'PROD_GA1X20',
'PROD_GA1X6');
spool off
exit
```

40 In one embodiment, a load\_template.sh file referenced by auto.ksh includes:

```
#!/usr/bin/sh
unalias cp
unalias rm
45 unalias mv
echo " Please wait; processing ADD_TEMPLATE_HERE"
LOG=$TOP/log/load_$LOG_NAME`date +%m%d%y`.log; export LOG
```

```

SQL_LOG=$TOP/log/sqlloader_$LOG_NAME`date +%m%d%y`.log; export
SQL_LOG
BAD_DATA=$TOP/bad/sqlloader_$LOG_NAME`date +%m%d%y`.bad; export
BAD_DATA
5 PATH=$ORACLE_HOME/bin:/bin:/usr/bin:/etc; export PATH
LD_LIBRARY_PATH=$ORACLE_HOME/lib; export LD_LIBRARY_PATH

echo "" >> $LOG 2>&1
echo "" >> $LOG 2>&1
10 echo "TOP.....: " $TOP >> $LOG 2>&1
echo "FILE_NAME.....: " $FILE_NAME >> $LOG
2>&1
echo "ORACLE_HOME.....: " $ORACLE_HOME >> $LOG
2>&1
15 echo "ORACLE_SID.....: " $ORACLE_SID >> $LOG
2>&1
echo "USERID.....: " $USERID >> $LOG 2>&1
echo "PASSWORD.....: " $PASS >> $LOG 2>&1
echo "Family Name.....: " $FAM_NAME >> $LOG 2>&1
20 echo "UI Flag.....: " $UI_FLAG >> $LOG 2>&1
echo "Status Enum.....: " $STATUS_ENUM >> $LOG
2>&1
echo "" >> $LOG 2>&1
echo "" >> $LOG 2>&1
25
sqlldr \
    userid=$USERID/$PASS@$ORACLE_SID \
    data=$TOP/data/$FILE_NAME \
    control=$TOP/ctl/ADD_TEMPLATE_HERE.ctl \
    log=$SQL_LOG \
    bad=$BAD_DATA \
    errors=100000 \
    bindsize=50000 \
    direct=y >> $LOG 2>&1
30
35
echo "" >> $LOG 2>&1
cat $SQL_LOG >> $LOG
rm -f $SQL_LOG
echo "" >> $LOG 2>&1
40
$ORACLE_HOME/bin/sqlplus -s $USERID/$PASS@$ORACLE_SID
<<EOF_RUN_TAG >> $LOG 2>&1
set pages 9999
SET SCAN OFF
45 set serveroutput on size 1000000
set timing on
prompt Executing map_ADD_TEMPLATE_HERE
exec map_ADD_TEMPLATE_HERE($UI_FLAG,$STATUS_ENUM);
prompt committing.
50 commit;
prompt You are done.

```

```
exit
EOF_RUN_TAG
```

```
exit 0
```

5

In one embodiment, a load\_setup\_IND.sh file for vertical IND referenced by auto.ksh includes text such as the following excerpt:

```
auto.ksh pvball 1085 1057 ind a pickone
10 auto.ksh pvbfly 1273 1246 ind a pickone
auto.ksh pvcheck 1274 1247 ind a pickone
auto.ksh pvdiaph 1275 1248 ind a pickone
auto.ksh pvgate 1276 1249 ind a pickone
auto.ksh pvglobe 1277 1250 ind a pickone
15 auto.ksh pvneedle 1278 1251 ind a pickone
auto.ksh pvothe 1279 1252 ind a pickone
```

In one embodiment, a map\_template.sql file referenced by auto.ksh includes:

20

```
--*****
--  

-- Name      : ADD_TEMPLATE_HERE.sql  

25 -- Usage    : This script is executed by  

load_ADD_TEMPLATE_HERE.sh script.  

-- Description: The following script would move data from the  

ADD_TEMPLATE_HERE temp  

--                  table to the several tables on the pims  

30 database.  

--*****/  

CREATE OR REPLACE PROCEDURE map_ADD_TEMPLATE_HERE (p_ui_flag  

integer default 0,p_status_enum integer default 0) AS  

35 l_record      ADD_TEMPLATE_HERE%ROWTYPE;  

lv_err_text      varchar2(81);  

lv_err_num       number;  

lv_user         varchar2(30);  

lv_prod_att_def_id number(5) := ADD_ATT_VAL_HERE;  

40 lv_defalut_node_id number(5) := ADD_NODE_VAL_HERE;  

lv_product_id   number     := 0;  

lv_sku_id       number     := 0;  

lv_flag          number     := 0;  

lv_count         number     := 0;  

45 lv_version     number     := 1;  

lv_text_id       text_data.text_id%TYPE := 0;  

lv_supplier_id   number     := 0;  

lv_related_product RELATED_PRODUCT.RELATED_PRODUCT%TYPE;  

p_errnum        number;  

50 p_errmsg       varchar2(81);
```

```

lv_check      number;
lv_buffer      varchar2(4000) := null;
lv_prod_desc_direction  varchar2(3) := null;
lv_clob_file_name   varchar(250) := null;
5   lv_blob_file_name   varchar(250) := null;
p_text_data_version text_data.text_data_version%TYPE := 1 ;
p_blob_data_version text_data.text_data_version%TYPE := 1 ;
lv_file_loc_1      number;
lv_success         varchar2(3)  :='yes';
10  lv_desc_dir_alias    varchar(30)  := null;
lv_blob_dir_alias    varchar(30)  := null;
lv_file_path        varchar(250) := null;
lv_file_tmp          varchar(250) := null;
lv_session_id        number := 0;
15  lv_sqlcode          number := 0;
lv_orig_mfg_count    number := 0;
lv_short_name        vendor.vendor_short_name%TYPE := null;

CURSOR c_get_all IS
20  SELECT *
     FROM ADD_TEMPLATE_HERE;

cursor get_orig_mfg_entry is select distinct ORIG_MFG_ID,
supplier_id
25  from ADD_TEMPLATE_HERE
where STATE_ENUM = 4 or STATE_ENUM is null ;

BEGIN

30  dbms_output.put_line('Moving data from the ADD_TEMPLATE_HERE
table to new schema.');

      SELECT user
      INTO lv_user
35  FROM dual;

      select userenv('SESSIONID') into lv_session_id FROM DUAL;

      OPEN c_get_all;

40  LOOP
      FETCH c_get_all INTO l_record;
      EXIT WHEN c_get_all%NOTFOUND;

45  lv_text_id      := null;
      lv_product_id := 0;
      lv_sku_id      := 0;
      lv_clob_file_name := null;
      lv_blob_file_name := null;
50  begin

```

```

    IF (l_record.SUPPLIER_ID is not null ) then
        lv_check := to_number(l_record.SUPPLIER_ID);
    ELSE
        dbms_output.put_line('ERROR sku_id=' || lv_sku_id
5          || ' was not inserted; supplier ID is NULL.');
        lv_success := 'no';
        GOTO end_of_loop;
    END IF;
EXCEPTION
10    WHEN OTHERS THEN
        lv_err_text := SUBSTR(SQLERRM,1,80);
        dbms_output.put_line('ERROR sku_id=' || lv_sku_id
            || ' was not inserted; supplier ID is not a number.');
        lv_success := 'no';
15    GOTO end_of_loop;
END;

lv_file_loc_1 := 0 ;

20    IF (l_record.PROD_DESC_DATA IS NOT NULL) then
        lv_prod_desc_direction :=
substr(l_record.PROD_DESC_DATA,1,1);
        IF lv_prod_desc_direction = '/' THEN
            lv_file_loc_1 := INSTR(l_record.PROD_DESC_DATA, '/',
25 -1, 1);
            lv_file_path := ' ' || substr(l_record.PROD_DESC_DATA,1,lv_file_loc_1 - 1);
            lv_file_path := lv_file_path || ' ';
        END IF;
        lv_clob_file_name :=
30    substr(l_record.PROD_DESC_DATA,lv_file_loc_1+1);
        --dbms_output.put_line ('Directory full path : ' ||
        lv_file_path);
        --dbms_output.put_line ('File name           : ' ||
35    lv_clob_file_name);

        lv_desc_dir_alias := 'ADD_TEMPLATE_HERE' ||
        lv_session_id || 'clob_dir';
        --dbms_output.put_line ('Directory alias is : ' ||
40    lv_desc_dir_alias );

        BEGIN
            EXECUTE IMMEDIATE 'drop      directory ' ||
lv_desc_dir_alias;
45        EXCEPTION
            WHEN OTHERS THEN
                null;
        END;

50    SAVEPOINT my_save_p;

```

```

        EXECUTE IMMEDIATE 'create directory ' ||
5      lv_desc_dir_alias || ' as ' || lv_file_path;
          text_data_clob (
            lv_clob_file_name,
            p_text_data_version,
            'ADD TEMPLATE HERE',
            'f',
            lv_session_id,
            lv_text_id,
10     p_errnum,
            p_errmsg);

        IF ( p_errnum != 0 ) THEN
          dbms_output.put_line('ERROR Sku_id=' ||
15    lv_sku_id
              || ' was not inserted. Error loading text
file='
              || l_record.PROD_DESC_DATA );
          dbms_output.put_line('Error num: ' ||
20    p_errnum||p_errmsg);

          rollback to my_save_p;
          lv_success := 'no';
          GOTO end_of_loop;
25      END IF;
      ELSE
        SAVEPOINT my_save_p;
          text_data_clob (
            l_record.PROD_DESC_DATA,
30      p_text_data_version,
            'ADD TEMPLATE HERE',
            't',
            lv_session_id,
            lv_text_id,
35      p_errnum,
            p_errmsg);

        IF ( p_errnum != 0 ) THEN
          dbms_output.put_line('ERROR inserting prod_desc:
40    Sku=' || lv_sku_id);
          dbms_output.put_line('Error num: ' ||
p_errnum||p_errmsg);

          rollback to my_save_p;
45      lv_success := 'no';
          GOTO end_of_loop;
        END IF;
      END IF;
END IF;

50  IF ( l_record.related_product IS NULL ) THEN

```

```
    lv_related_product := 'Unspecified';
else
    lv_related_product := l_record.related_product;
end if;
5
SELECT product_id.NextVal
INTO  lv_product_id
FROM dual;

10      select sku_id.NextVal
       into lv_sku_id
       from dual;

BEGIN
15      INSERT INTO product (
        PRODUCT_ID,
        PROD_ATT_DEF_ID,
        DEFAULT_NODE_ID,
        CREATE_DATE,
        CREATED_BY,
        ORIG_MFG_ID,
        PROD_NAME,
        PROD_DESC_ID,
        SUPPLIER_ID,
        REFERENCE,
        SYNONYMS,
        PROD_GA500X1,
        APPLICATION,
        PROD_GA500X2,
        PROD_GA500X3,
        SHIPPING_TYPE_ENUM,
        ADD_INFO,
        PROD_GA1X1,
        PROD_GA1X2,
        PROD_GA1X3,
        PROD_GA500X4,
        PROD_GA250X1,
        PROD_GA1X4,
        PROD_GA250X2,
        PROD_GA1X5,
        PROD_GA1X6,
        REVIEW_ENUM,
        AGENCY_ENUM,
        REGULATORY_SCHEDULE_ENUM,
        REGULATORY_ACTION_ENUM,
        Modified_date,
        Modified_by,
        SUPPLIER_PRODUCT_NAME,
        unspsc_code,
        status_enum
        ADD_PROD_GA_HERE)
20
25
30
35
40
45
50
```

```

      VALUES (
      lv_product_id,
      l_record.PROD_ATT_DEF_ID,
      l_record.DEFAULT_NODE_ID,
      sysdate,
      lv_user,
      l_record.ORIG_MFG_ID,
      l_record.PROD_NAME,
      lv_text_id,
      l_record.SUPPLIER_ID,
      l_record.REFERENCE,
      l_record.SYNONYMS,
      l_record.PROD_GA500X1,
      l_record.APPLICATION,
      l_record.PROD_GA500X2,
      l_record.PROD_GA500X3,
      l_record.SHIPPING_TYPE,
      l_record.ADD_INFO,
      l_record.PROD_GA1X1,
      l_record.PROD_GA1X2,
      l_record.PROD_GA1X3,
      l_record.PROD_GA500X4,
      l_record.PROD_GA250X1,
      l_record.PROD_GA1X4,
      l_record.PROD_GA250X2,
      l_record.PROD_GA1X5,
      l_record.PROD_GA1X6,
      l_record.REVIEW_ENUM,
      l_record.AGENCY_ENUM,
      l_record.REGULATORY_SCHEDULE_ENUM,
      l_record.REGULATORY_ACTION_ENUM,
      sysdate,
      'PIMS_LOAD',
      l_record.SUPPLIER_PRODUCT_NAME,
      l_record.unspsc_code,
      p_status_enum
      ADD_PROD_GA_VAL_HERE
    );

40      EXCEPTION
        WHEN OTHERS THEN
          lv_err_num := SQLCODE;
          lv_err_text := SUBSTR(SQLERRM,1,80);
          dbms_output.put_line('ERROR Sku_id=' || lv_sku_id || ' was
45      not inserted. Error inserting into the Product table.');
          dbms_output.put_line('Error #: ' ||
          lv_err_num||lv_err_text);

          rollback to my_save_p;
          lv_success := 'no';
50      GOTO end_of_loop;

```

```
END;

BEGIN

    5      INSERT INTO sku (
        SKU_ID,
        SKU_VERSION,
        PRODUCT_ID,
        CREATE_DATE,
        10     CREATED_BY,
        UNIT,
        QTY,
        UNIT_MULTIPLIER,
        WEIGHT,
        15     SHIP_WT,
        CMDX_SKU,
        SUPPLIER_CAT_NUM,
        BARCODE,
        STATE_ENUM,
        20     ORIG_MFG_CAT_NUM,
        Modified_date,
        modified_by,
        status_enum,
        splr_cat_number_differentiator,
        25     normalized_unit_multiplier,
        normalized_qty,
        normalized_unit)
        VALUES (
        lv_sku_id,
        30     lv_version,
        lv_product_id,
        sysdate,
        lv_user,
        l_record.unit,
        l_record.qty,
        35     nvl(l_record.unit_multiplier,1),
        l_record.weight,
        l_record.ship_wt,
        lv_sku_id,
        l_record.supplier_cat_num,
        40     l_record.barcode,
        l_record.STATE_ENUM,
        l_record.orig_mfg_cat_num,
        sysdate,
        'PIMS_LOAD',
        p_status_enum,
        45     null,
        l_record.normalized_unit_multiplier,
        l_record.normalized_qty,
        l_record.normalized_unit);
    50
```

```

EXCEPTION
    WHEN OTHERS THEN
        lv_err_num := SQLCODE;
        lv_err_text := SUBSTR(SQLERRM,1,80);
5
    begin
        select vendor_short_name
        into lv_short_name from vendor
        where vendor_id = l_record.SUPPLIER_ID
10       and vendor_short_name is not null;
    EXCEPTION
        WHEN OTHERS THEN
            null;
    END;
15
        dbms_output.put_line('ERROR inserting into the Sku
table; Sku = ' || lv_sku_id);
        dbms_output.put_line('Error #      : ' ||
lv_err_num||lv_err_text);
20        --dbms_output.put_line(' lv_sku_id      : ' ||
lv_sku_id);
        --dbms_output.put_line(' lv_product_id : ' ||
lv_product_id);
        dbms_output.put_line(' CMDX Sku      : ' ||
lv_sku_id);
25        dbms_output.put_line(' Vendor      : ' ||
lv_short_name);
        dbms_output.put_line(' supp cat num : ' ||
l_record.supplier_cat_num);
        dbms_output.put_line(' QTY      : ' ||
30 l_record.qty);
        dbms_output.put_line(' Unit      : ' ||
l_record.unit);
        dbms_output.put_line(' Unit Mul      : ' ||
NVL(l_record.unit_multiplier,1));
35        dbms_output.put_line('.');

        rollback to my_save_p;
        lv_success := 'no';
        GOTO end_of_loop;
40
    END;

BEGIN
    INSERT INTO price (
        PRICE_ID,
        BUYER_ID,
45        BUYER_SEG_ID,
        SUPPLIER_ID,
        SKU_ID,
        MIN_QTY,
50        PRICE_VERSION,
        CREATED_BY,

```

```

CREATE_DATE,
PRICE,
MODIFIED_BY,
MODIFIED_DATE,
STATE_ENUM,
STATUS_ENUM,
USE_LIST_DSCNT,
FROM_EFFECTIVE_DATE,
TO_EFFECTIVE_DATE,
LIST_DSCNT_PCT)
values (
    PRICE_ID.NEXTVAL,
0,
0,
15    l_record.SUPPLIER_ID,
lv_sku_id,
0,
1000000,
lv_user,
20    sysdate,
l_record.list_price,
'PIMS_LOAD',
sysdate,
l_record.state_enum,
p_status_enum,
25    null,
to_date('01-JAN-0001','DD-MON-YYYY'),
to_date('31-DEC-9999','DD-MON-YYYY'),
null);

30 EXCEPTION
    WHEN OTHERS THEN
        lv_err_num := SQLCODE;
        lv_err_text := SUBSTR(SQLERRM,1,80);
        dbms_output.put_line('ERROR Sku_id=' || lv_sku_id
35    || ' was not inserted. Error inserting unaffiliated
price sheet.');
        dbms_output.put_line('Error #: ' || lv_err_num);

        rollback to my_save_p;
40    lv_success := 'no';
        GOTO end_of_loop;
END;

BEGIN
45    INSERT INTO related_product (
        PRODUCT_ID,
        SORT_ORDER,
        REFERENCE_PROD_ID,
        REFERENCE_TYPE_ENUM,
50    RELATION_TYPE_ENUM,
        RELATED_PRODUCT,

```

```

        EXTERNAL_REFERENCE,
        STATUS_ENUM
    )
VALUES (
5      lv_product_id,
1,
null,
null,
null,
10     lv_related_product,
null,
p_status_enum
);

15    EXCEPTION
        WHEN OTHERS THEN
            lv_err_num := SQLCODE;
            lv_err_text := SUBSTR(SQLERRM,1,80);
            dbms_output.put_line('ERROR Sku_id=' || lv_sku_id
20           || ' was not inserted. Error inserting into Related
Product');
            dbms_output.put_line('Error #: ' || lv_err_num);

            rollback to my_save_p;
25            lv_success := 'no';
            GOTO end_of_loop;
        END;

/*
30    IF (l_record.blob_file_name IS NOT NULL) then

        mime_data_blob (
            lv_blob_file_name,
            p_blob_data_version,
35            'ADD_TEMPLATE_HERE',
            lv_session_id,
            lv_sku_id,
            p_errnum,
            p_errmsg);
40

        IF ( p_errnum != 0 ) THEN
            dbms_output.put_line('ERROR Sku_id=' || lv_sku_id
                || ' was not inserted. Error loading binary file='
                || l_record.blob_file_name);
45            dbms_output.put_line('Error num: ' || p_errnum);

            rollback to my_save_p;
            lv_success := 'no';
            GOTO end_of_loop;
50    END IF;

```

```

        END IF;
    */

<<end_of_loop>>

5      IF lv_success = 'yes' THEN
          lv_count := lv_count + 1;
      END IF;

10     lv_success := 'yes';

      lv_flag := mod(lv_count, 1000);
      IF ( (lv_flag = 0) and (lv_count > 0) ) THEN
          dbms_output.put_line('Committing at ' || lv_count || '.');
15     commit;
      END IF;

END LOOP;

20 CLOSE c_get_all;

FOR get_one_rec in get_orig_mfg_entry LOOP

      lv_orig_mfg_count := 0;
25      -- IF (get_one_rec.ORIG_MFG_ID is NULL ) THEN
          begin
              select count(SUPPLIER_ID)
                  into lv_orig_mfg_count
                  from ORIG_MFG
30                  where SUPPLIER_ID = get_one_rec.SUPPLIER_ID
                  and ORIG_MFG_ID = get_one_rec.ORIG_MFG_ID;

              IF (lv_orig_mfg_count = 0) THEN
                  insert into ORIG_MFG ( SUPPLIER_ID, ORIG_MFG_ID,
35      UI_FLAG)
                      values (get_one_rec.SUPPLIER_ID,
                  get_one_rec.SUPPLIER_ID, p_ui_flag);
              END IF;
              exception
40                  WHEN OTHERS THEN
                      lv_sqlcode := sqlcode;
                      if ( lv_sqlcode = -1 ) then
                          null; -- It is OK, record may already
exist.
45                  else
                      dbms_output.put_line('Error inserting into
Orig_Mfg table; please contact DBA group.');
                      end if;
                  end;
      -- ELSE
50      IF (get_one_rec.ORIG_MFG_ID is NOT NULL ) THEN

```

```

begin
    select count(SUPPLIER_ID)
        into lv_orig_mfg_count
        from ORIG_MFG
5      where SUPPLIER_ID = get_one_rec.SUPPLIER_ID
        and ORIG_MFG_ID = get_one_rec.ORIG_MFG_ID;

        IF (lv_orig_mfg_count = 0) THEN
            insert into ORIG_MFG ( SUPPLIER_ID, ORIG_MFG_ID,
10    UI_FLAG)
                values (get_one_rec.SUPPLIER_ID,
get_one_rec.ORIG_MFG_ID, p_ui_flag);
            END IF;
            exception
15        WHEN OTHERS THEN
            lv_sqlcode := sqlcode;
            if ( lv_sqlcode = -1 ) then
                null; -- It is OK, record may already
exist.
20        else
            dbms_output.put_line('Error inserting into
Orig_Mfg table; please contact DBA group.');
            end if;
        end;
25    END IF;

    END LOOP;

    dbms_output.put_line(lv_count || ' Products have been
30 inserted. ');

    commit;

-- Some clean up for all dir aliases;
35 BEGIN
        EXECUTE IMMEDIATE 'drop directory ' ||
lv_desc_dir_alias;
    EXCEPTION
        WHEN OTHERS THEN
30    null;
END;

BEGIN
40    EXECUTE IMMEDIATE 'drop directory ' ||
45    lv_blob_dir_alias;
    EXCEPTION
        WHEN OTHERS THEN
        null;
END;
50    EXCEPTION

```

```
WHEN OTHERS THEN
    lv_err_text := SUBSTR(SQLERRM,1,80);
    dbms_output.put_line('Error map_ADD_TEMPLATE_HERE Main : '
5   || lv_err_text);
    rollback;
END map_ADD_TEMPLATE_HERE;
/
show errors;
exit;
```

In one embodiment, a template.ctl file referenced by auto.ksh includes:

```
load data
replace
15  into table ADD_TEMPLATE_HERE
FIELDS TERMINATED BY "|"
TRAILING NULLCOLS
(
20    cmdx_sku_id ,
    PROD_ATT_DEF_ID,
    DEFAULT_NODE_ID,
    Orig_Mfg_Cat_Num ,
    Orig_Mfg_Id ,
    Prod_Name ,
    prod_desc_data ,
    Supplier_Id ,
    Weight ,
    Barcode ,
    List_Price ,
    Supplier_Cat_Num ,
30    Qty ,
    Unit ,
    Unit_Multiplier ,
    STATE_ENUM,
    reference ,
    related_product ,
    Synonyms ,
    prod_GA500X1 ,
    application ,
    Prod_GA500X2 ,
    Prod_GA500X3 ,
    Ship_Wt ,
    Shipping_Type ,
    add_info ,
    Prod_GA1X1 ,
    Prod_GA1X2 ,
    Prod_GA1X3 ,
    Prod_GA500X4 ,
    Prod_GA250X1 ,
    Prod_GA1X4 ,
    Prod_GA250X2 ,
```

```

Prod_GA1X5 ,
Prod_GA1X6 ,
REVIEW_ENUM ,
AGENCY_ENUM ,
5 REGULATORY_SCHEDULE_ENUM,
REGULATORY_ACTION_ENUM,
SUPPLIER_PRODUCT_NAME,
UNSPSC_CODE,
NORMALIZED_UNIT_MULTIPLIER,
10 NORMALIZED_QTY,
NORMALIZED_UNIT
ADD_GA_HERE
)

```

15

In one embodiment, a template.sql file referenced by auto.ksh includes:

```

drop table ADD_TEMPLATE_HERE cascade constraints;

20 CREATE TABLE ADD_TEMPLATE_HERE (
    CMDX_Sku_Id          varchar2(120) NOT NULL,
    prod_Att_Def_id      number(12) NOT NULL,
    default_Node_ID       number(12) NOT NULL,
    Orig_Mfg_Cat_Num     VARCHAR2(64) NULL,
    Orig_Mfg_Id           NUMBER(12) NULL,
    Prod_Name              VARCHAR2(500) NULL,
    prod_desc_data        VARCHAR2(4000) NULL,
    Supplier_Id            varchar2(120) NULL,
    Weight                 NUMBER(15,5) NULL,
    30 Barcode                NUMBER(20) NULL,
    List_Price              NUMBER(10,2) NULL,
    Supplier_Cat_Num       VARCHAR2(64) NULL,
    Qty                     NUMBER NULL,
    Unit                    VARCHAR2(20) NULL,
    35 Unit_Multiplier       NUMBER(8) NULL,
    state_enum               NUMBER(5) NULL,
    reference                VARCHAR2(4000) NULL,
    related_product          VARCHAR2(500) NULL,
    Synonyms                  VARCHAR2(2000) NULL,
    40 prod_GA500X1            VARCHAR2(500) NULL,
    application               VARCHAR2(4000) NULL,
    Prod_GA500X2              VARCHAR2(500) NULL,
    Prod_GA500X3              VARCHAR2(500) NULL,
    Ship_Wt                  NUMBER(15,5) NULL,
    45 Shipping_Type            NUMBER(12) NULL,
    add_info                  VARCHAR2(4000) NULL,
    Prod_GA1X1                  CHAR(1) NULL,
    Prod_GA1X2                  CHAR(1) NULL,
    Prod_GA1X3                  CHAR(1) NULL,
    50 Prod_GA500X4            VARCHAR2(500) NULL,
    Prod_GA250X1              VARCHAR2(250) NULL,

```

```

Prod_GA1X4           CHAR(1) NULL,
Prod_GA250X2         VARCHAR2(250) NULL,
Prod_GA1X5           CHAR(1) NULL,
Prod_GA1X6           CHAR(1) NULL,
5                  REVIEW_ENUM      NUMBER(5) NULL,
AGENCY_ENUM          NUMBER(5) NULL,
REGULATORY_SCHEDULE_ENUM NUMBER(5) NULL,
REGULATORY_ACTION_ENUM NUMBER(5) NULL,
supplier_product_name VARCHAR2(500),
10                 unspsc_code    VARCHAR2(50),
normalized_unit_multiplier NUMBER,
normalized_qty        NUMBER,
normalized_unit       NUMBER
ADD_GA_HERE

15                );
quit

```

In one embodiment, the load\_setup.out script invokes the auto.ksh script, which uses various files together with the ontology values passed as parameters by load\_setup.out script in order to produce engineering configuration files.

## Summary

In summary, the present invention provides tools and techniques for automatically generating marketplace web site configuration files and other configuration materials (e.g., supplier documentation) from an express structural ontology specification. The structural ontology specification 406 can expressly specify a data type, and perhaps also a data size, for at least one attribute using, e.g., a “Data type” column in a spreadsheet 804. The specification 406 can expressly specify allowed data values for at least one attribute using, e.g., an “Allowed values (for ENUMS)” column in such a spreadsheet 804. The specification 406 can expressly specify a search type for at least one attribute using, e.g., a “Search method” column in such a spreadsheet 804. The specification 406 can expressly specify that at least one attribute is mandatory or that it is optional using, e.g., Y or N values respectively in a “Required? (Y or N)” column in such a spreadsheet 804. The specification 406 can expressly specify a mapping between an attribute and a product relational database field using, e.g., PIMS Mapping and Att\_Def\_Id columns in a such spreadsheet 804. It may also specify other ontological characteristics of the marketplace, but the focus here is on structural ontology rather than transactional ontology.

The invention does not solve all configuration problems of market site providers. Even when scripts are used to generate 506 configuration files from an express structural ontology specification 406, bugs that need fixing may occur. For example, if an ontology.properties file

is changed but corresponding radio.properties and enum.properties files are not updated as well, then problems may occur. In one such case, enum file order sequencing incorrectly counted text-boxes as drop-downs when a text-box preceded a drop-down, which threw off the enum display listings. In addition, problems may occur as a result of human errors when  
5 creating or updating the express structural ontology specification 406 itself. Nonetheless, the invention facilitates marketplace configuration by providing greater consistency and ease of revision in some cases. In addition to their consistency with an authoritative ontology specification 406 and their relative ease of generation once the invention is implemented, automatically generated configuration files 808 can provide other advantages over manually  
10 produced configuration files. For instance, automatically generated prod\_att\_def\_id files can be easily produced in sorted form.

In general, file excerpts shown above are not exclusive, in that embodiments of the invention may have files containing additional text; words such as "contains" and "includes" should be understood accordingly to mean "comprises". There may also be some duplication,  
15 such that information appears in more than one of the applications (namely, the three incorporated provisional applications or the present application) or more than once in a particular application. Extraneous information, blank lines, and other white space have sometimes been removed, fonts have been changed, and other formatting steps were taken.

More generally, it should also be noted that the applications may contain more detail  
20 than is required by law in a patent. U.S. patent case law indicates, for example, that computer source code is not necessarily required in an application, but several script source codes are included in the applications. These implementation details are provided in order to err – if errors are being made – by including too much information rather than including too little. Applicants should not be penalized for being so forthcoming. In particular, the inclusion of  
25 details in an application should not be viewed as an assumption or admission that those details, or similar details, or a similar level of detail, are actually required to support the claims ultimately granted. Nor should the inclusion of particular implementation details be misinterpreted by treating as inventors people who simply implemented inventive ideas conceived by others.

30 Articles of manufacture within the scope of the present invention include a computer-readable storage medium in combination with the specific physical configuration of a substrate of the computer-readable storage medium. The substrate configuration represents data and instructions which cause the computers to operate in a specific and predefined manner as described herein. Suitable storage devices include hard disks, CD-ROMs, and other media

readable by computers. Each such medium tangibly embodies a program that is executable by a computer system to perform steps substantially as described herein to use an express structural ontology specification of a marketplace as a basis for automatically generating configuration materials for the marketplace.

5        Although particular methods embodying the present invention are expressly illustrated and described herein, it will be appreciated that system and configured storage medium embodiments may also be formed according to the methods of the present invention. Unless otherwise expressly indicated, the descriptions herein of methods of the present invention therefore extend to corresponding systems and configured storage media, and the descriptions  
10      of systems and configured storage media of the present invention extend likewise to corresponding methods. An “embodiment” of the invention may be, without limitation, a system, an article of manufacture, a method, and/or a computer memory, CD, disk, or other digital or analog medium that is configured according to the invention. The method steps may be performed in various orders, except in those cases in which the results of one step are  
15      required as input to another step. Likewise, steps may be omitted unless called for in issued claims, regardless of whether they are expressly described as optional in this Detailed Description. Steps may also be repeated, combined, or named differently.

All claims as filed are part of the specification and thus help describe the invention, and claim language may be repeated outside the claims as needed. As used herein, terms such as  
20      “a” and “the” and designations such as “file”, “attribute”, and “script” are inclusive of one or more of the indicated element. In particular, in the claims a reference to an element normally means at least one such element is required.

The invention may be embodied in other specific forms without departing from its essential characteristics. The described embodiments are to be considered in all respects only  
25      as illustrative and not restrictive. Headings are for convenience only. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by patent is:

## CLAIMS

1. A method for facilitating electronic commerce, comprising the steps of:
  - (a) obtaining an express structural ontology specification for a particular electronic commerce marketplace, the structural ontology specification:
    - 5 (i) being organized in a predefined format so that it can be parsed by a computer,
    - (ii) being an express and hence human-readable specification rather than being merely implicit in computer program code, and
    - (iii) expressly specifying at least product categories, product generic attributes, and product category attributes;
  - (b) automatically parsing the structural ontology specification using a computerized tool which also parses other structural ontology specifications written in the predefined format;
  - (c) extracting ontology information from the structural ontology specification using a computerized tool which also extracts ontology information from other structural ontology specifications; and
  - (d) using ontology information extracted from the structural ontology specification to automatically generate for the electronic commerce marketplace configuration materials, at least some of which configuration materials are not product catalog files.

2. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate configuration files that include at least three of:

- a user interface configuration file,
- 25 a search interface configuration file,
- a file containing a user interface quality assurance checklist,
- a file containing a search interface quality assurance checklist,
- a file containing a framework of a script for extracting product data from a text file and loading the product data into a product relational database,
- 30 a file containing a product data quality assurance script, and
- product catalog files.

3. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate configuration files that include at least four of:

a user interface configuration file,  
a search interface configuration file,  
a file containing a user interface quality assurance checklist,  
a file containing a search interface quality assurance checklist,  
5 a file containing a framework of a script for extracting product data from a text  
file and loading the product data into a product relational database,  
a file containing a product data quality assurance script, and  
product catalog files.

4. The method of claim 1, wherein the method uses ontology information extracted

10 from the structural ontology specification to automatically generate configuration files that  
include at least five of:

a user interface configuration file,  
a search interface configuration file,  
a file containing a user interface quality assurance checklist,  
a file containing a search interface quality assurance checklist,  
15 a file containing a framework of a script for extracting product data from a text  
file and loading the product data into a product relational database,  
a file containing a product data quality assurance script, and  
product catalog files.

5. The method of claim 1, wherein the method uses ontology information extracted

from the structural ontology specification to automatically generate configuration files that  
include at least six of:

a user interface configuration file,  
a search interface configuration file,  
25 a file containing a user interface quality assurance checklist,  
a file containing a search interface quality assurance checklist,  
a file containing a framework of a script for extracting product data from a text  
file and loading the product data into a product relational database,  
a file containing a product data quality assurance script, and  
product catalog files.

6. The method of claim 1, wherein the structural ontology specification expressly  
specifies a data type for at least one attribute.

7. The method of claim 1, wherein the structural ontology specification expressly  
specifies a data size for at least one attribute.

8. The method of claim 1, wherein the structural ontology specification expressly specifies allowed data values for at least one attribute.

9. The method of claim 1, wherein the structural ontology specification expressly specifies a search type for at least one attribute.

5 10. The method of claim 1, wherein the structural ontology specification expressly specifies that at least one attribute is mandatory.

11. The method of claim 1, wherein the structural ontology specification expressly specifies that at least one attribute is optional.

10 12. The method of claim 1, wherein the structural ontology specification expressly specifies a mapping between an attribute and a product relational database field.

13. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one user interface configuration file.

15 14. The method of claim 13, wherein the method automatically generates a user interface search configuration file.

15. The method of claim 13, wherein the method automatically generates a user interface product display configuration file.

20 16. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one search interface configuration file.

17. The method of claim 16, wherein the method automatically generates a search interface initial database request configuration file.

18. The method of claim 16, wherein the method automatically generates a search interface product detail request interface configuration file.

25 19. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one catalog configuration file.

20. The method of claim 19, wherein the method automatically generates a catalog mapping sheet configuration file.

30 21. The method of claim 19, wherein the method automatically generates a catalog enumeration load sheet configuration file.

22. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a user interface quality assurance checklist.

23. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a search interface quality assurance checklist.

24. The method of claim 1, wherein the method uses ontology information extracted  
5 from the structural ontology specification to automatically generate at least one file containing a framework of a script for extracting product data from a text file.

25. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a framework of a script for loading product data into a product relational database.

10 26. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a product data quality assurance script.

15 27. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one configuration file for a graphical product data entry tool which accepts product data entered manually by a user and places the product data in a product relational database.

20 28. The method of claim 1, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing documentation which describes product data that is requested from a supplier regarding products to be offered in the electronic commerce marketplace.

29. A computer system, comprising:

(a) an express structural ontology specification for a particular electronic commerce marketplace, the structural ontology specification:

(i) being organized in a predefined format so that it can be parsed by a computer  
25 in the system,

(ii) being an express and hence human-readable specification rather than being merely implicit in computer program code, and

(iii) expressly specifying at least product categories, product generic attributes,  
and product category attributes;

30 and

(b) a computerized file generation means for parsing the structural ontology specification, extracting ontology information from the structural ontology specification, and using the ontology information to automatically generate for the

electronic commerce marketplace configuration materials, at least some of which configuration materials are not product catalog files.

30. The system of claim 29, wherein the file generation means automatically generates configuration files that include at least two of:

- 5 a user interface configuration file,
- a search interface configuration file,
- a file containing a user interface quality assurance checklist,
- a file containing a search interface quality assurance checklist,
- a file containing a framework of a script for extracting product data from a text  
10 file,
- a file containing a framework of a script for loading the product data into a product relational database,
- a file containing a product data quality assurance script,
- a configuration file for a graphical product data entry tool,
- 15 a file containing documentation which describes product data that is requested from a supplier, and
- a catalog configuration file.

31. The system of claim 29, wherein the file generation means automatically generates configuration files that include at least four of:

- 20 a user interface configuration file,
- a search interface configuration file,
- a file containing a user interface quality assurance checklist,
- a file containing a search interface quality assurance checklist,
- a file containing a framework of a script for extracting product data from a text  
25 file,
- a file containing a framework of a script for loading the product data into a product relational database,
- a file containing a product data quality assurance script,
- a configuration file for a graphical product data entry tool,
- 30 a file containing documentation which describes product data that is requested from a supplier, and
- a catalog configuration file.

32. The system of claim 29, wherein the file generation means automatically generates configuration files that include at least six of:

a user interface configuration file,  
a search interface configuration file,  
a file containing a user interface quality assurance checklist,  
a file containing a search interface quality assurance checklist,  
5 a file containing a framework of a script for extracting product data from a text  
file,

a file containing a framework of a script for loading the product data into a  
product relational database,

a file containing a product data quality assurance script,

10 a configuration file for a graphical product data entry tool,

a file containing documentation which describes product data that is requested  
from a supplier, and

a catalog configuration file.

33. The system of claim 29, wherein the file generation means automatically

15 generates configuration files that include at least eight of:

a user interface configuration file,

a search interface configuration file,

a file containing a user interface quality assurance checklist,

a file containing a search interface quality assurance checklist,

20 a file containing a framework of a script for extracting product data from a text  
file,

a file containing a framework of a script for loading the product data into a  
product relational database,

a file containing a product data quality assurance script,

25 a configuration file for a graphical product data entry tool,

a file containing documentation which describes product data that is requested  
from a supplier, and

a catalog configuration file.

34. The system of claim 29, wherein the structural ontology specification expressly

30 specifies at least one of: data type for at least one attribute, and a data size for at least one  
attribute.

35. The system of claim 29, wherein the structural ontology specification expressly  
specifies allowed data values for at least one attribute.

36. The system of claim 29, wherein the structural ontology specification expressly specifies a search type for at least one attribute.

37. The system of claim 29, wherein the structural ontology specification expressly specifies that at least one attribute is mandatory and/or specifies that at least one attribute is optional.

38. The system of claim 29, wherein the structural ontology specification expressly specifies a mapping between an attribute and a product relational database field.

39. The system of claim 29, wherein the file generation means comprises a script.

40. The system of claim 29, wherein the structural ontology specification comprises a spreadsheet file.

41. A computer system, comprising:

a processor;

memory accessible to the processor configured by an express structural ontology specification for a particular electronic commerce marketplace; and

software for the processor which uses ontology information read from the structural ontology specification to generate for the electronic commerce marketplace configuration materials, at least some of which configuration materials are not product catalog files.

42. The system of claim 41, in which the software comprises a script.

43. The system of claim 41, in which the software comprises at least two scripts, and one script invokes the other script.

44. The system of claim 41, in which the software generates interface configuration files.

45. The system of claim 41, in which the software generates catalog configuration files.

46. The system of claim 41, in which the software generates quality assurance files.

47. The system of claim 41, in which the software generates supplier documentation files.

48. The system of claim 41, in which the software generates data extraction files.

49. The system of claim 41, in which the software generates database load files.

50. A computer-readable storage media configured to perform a method comprising the steps of parsing an express structural ontology specification, extracting ontology information from the structural ontology specification, and using the ontology information to

automatically generate for the electronic commerce marketplace configuration materials, at least some of which configuration materials are not product catalog files.

51. The configured medium of claim 50, wherein the method automatically generates a user interface search configuration file.

5 52. The configured medium of claim 50, wherein the method automatically generates a user interface product display configuration file.

53. The configured medium of claim 50, wherein the method automatically generates a search interface initial database request configuration file.

10 54. The configured medium of claim 50, wherein the method automatically generates a search interface product detail request interface configuration file.

55. The configured medium of claim 50, wherein the method automatically generates a catalog mapping sheet configuration file.

56. The configured medium of claim 50, wherein the method automatically generates a catalog enumeration load sheet configuration file.

15 57. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a user interface quality assurance checklist.

20 58. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a search interface quality assurance checklist.

59. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a framework of a script for extracting product data from a text file.

25 60. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a framework of a script for loading product data into a product relational database.

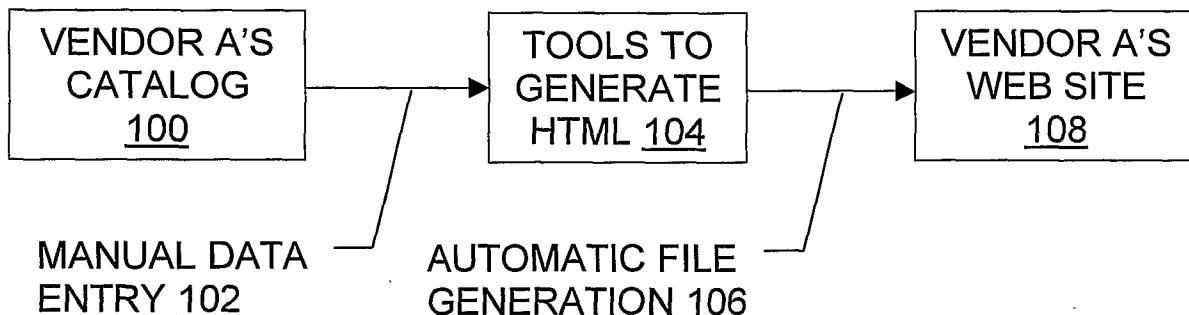
30 61. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing a product data quality assurance script.

62. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one configuration file for a graphical product data entry tool which accepts product data entered manually by a user and places the product data in a product relational database.

63. The configured medium of claim 50, wherein the method uses ontology information extracted from the structural ontology specification to automatically generate at least one file containing documentation which describes product data that is requested from a supplier regarding products to be offered in the electronic commerce marketplace.

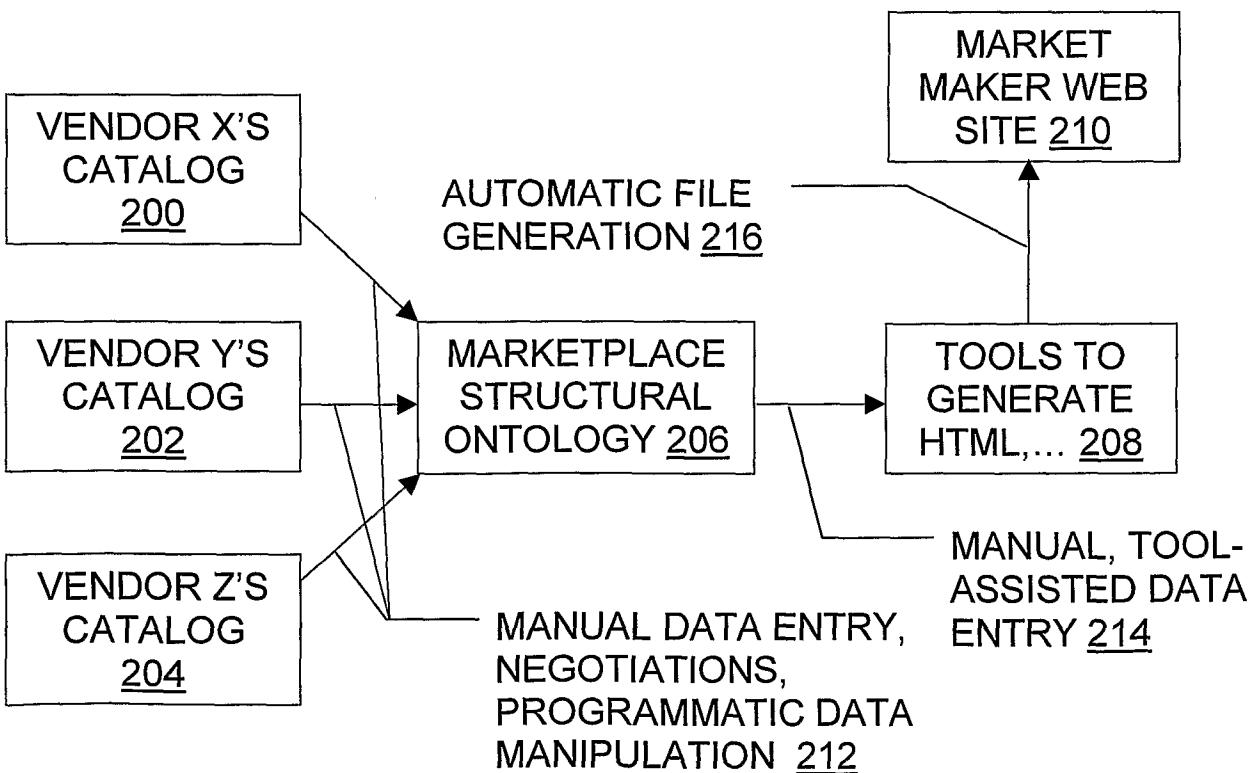
5

1/8



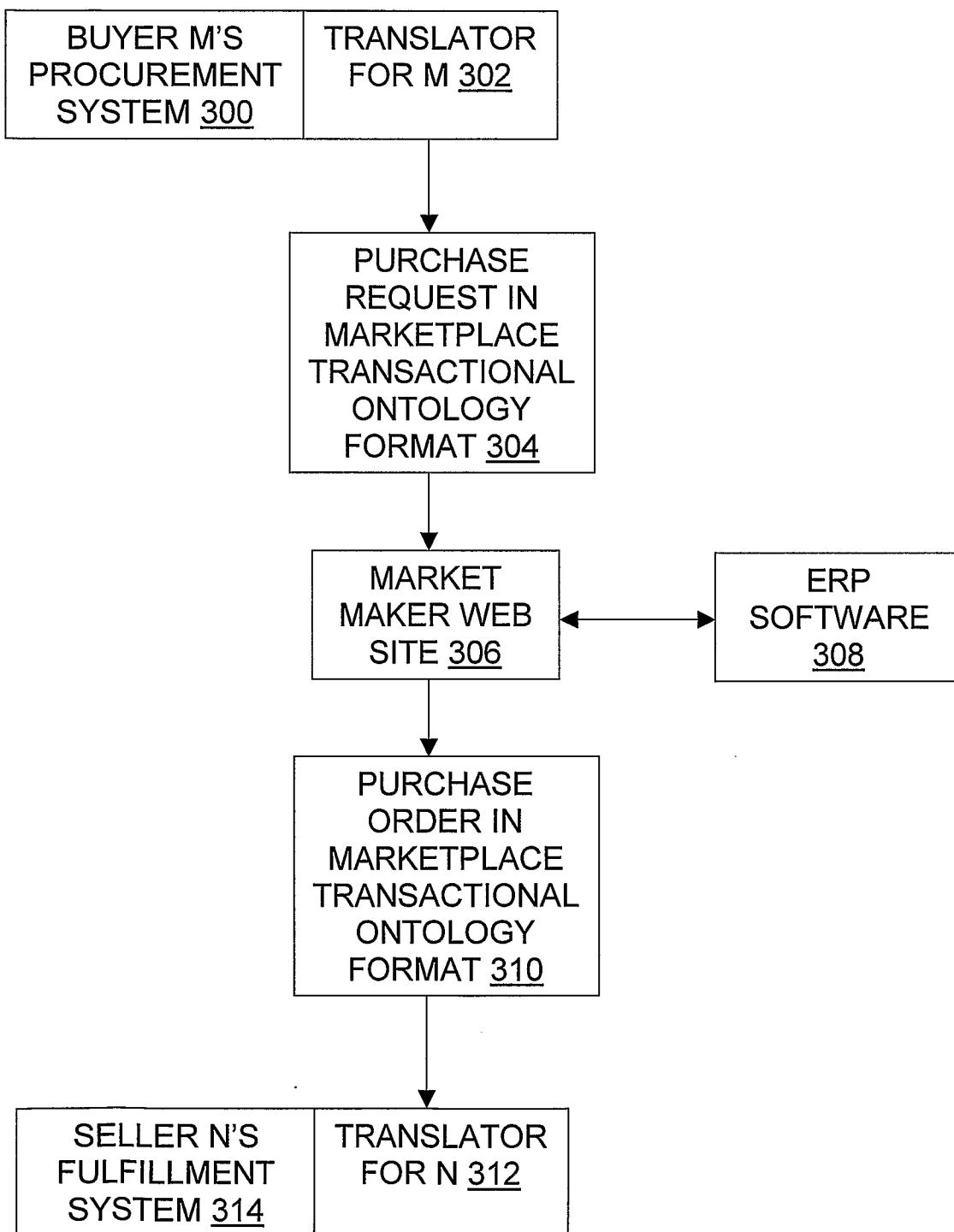
(PRIOR ART)

Fig. 1



(PRIOR ART)

Fig. 2



(PRIOR ART)

Fig. 3

3/8

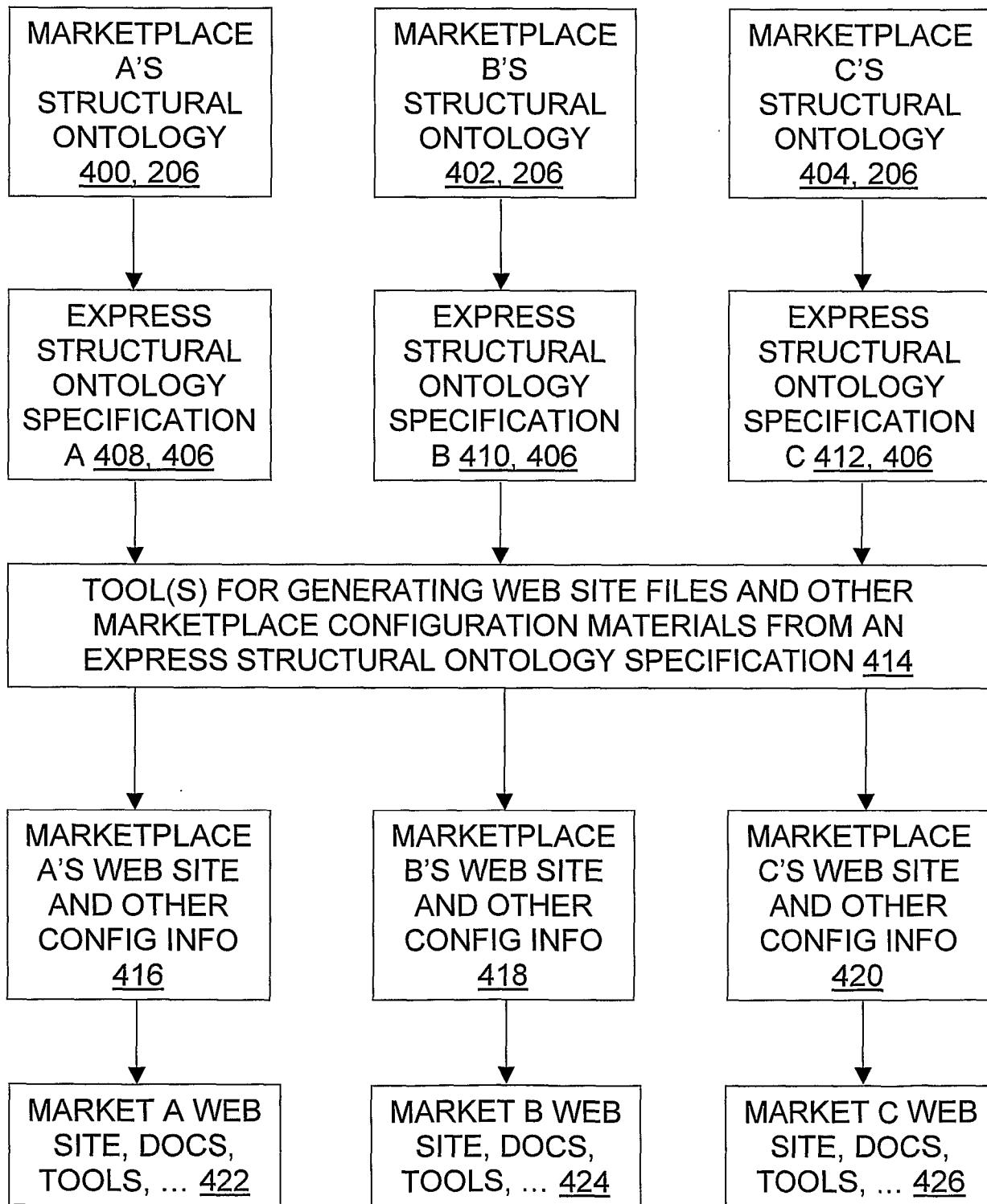


Fig. 4

4/8

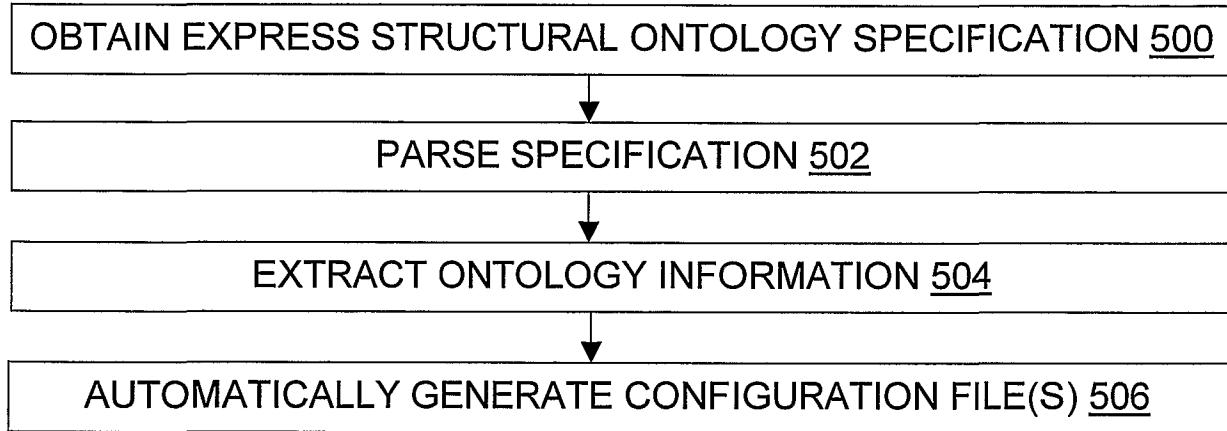


Fig. 5

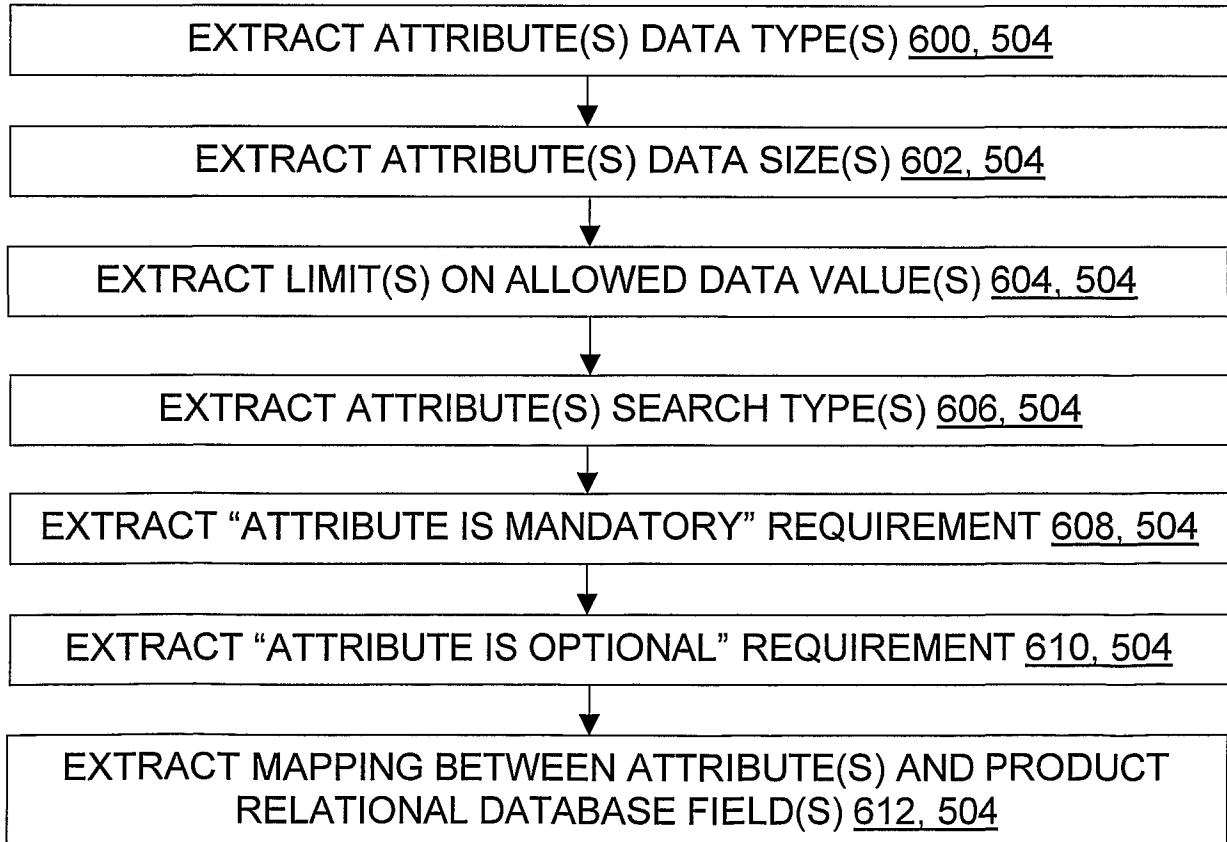


Fig. 6

5/8

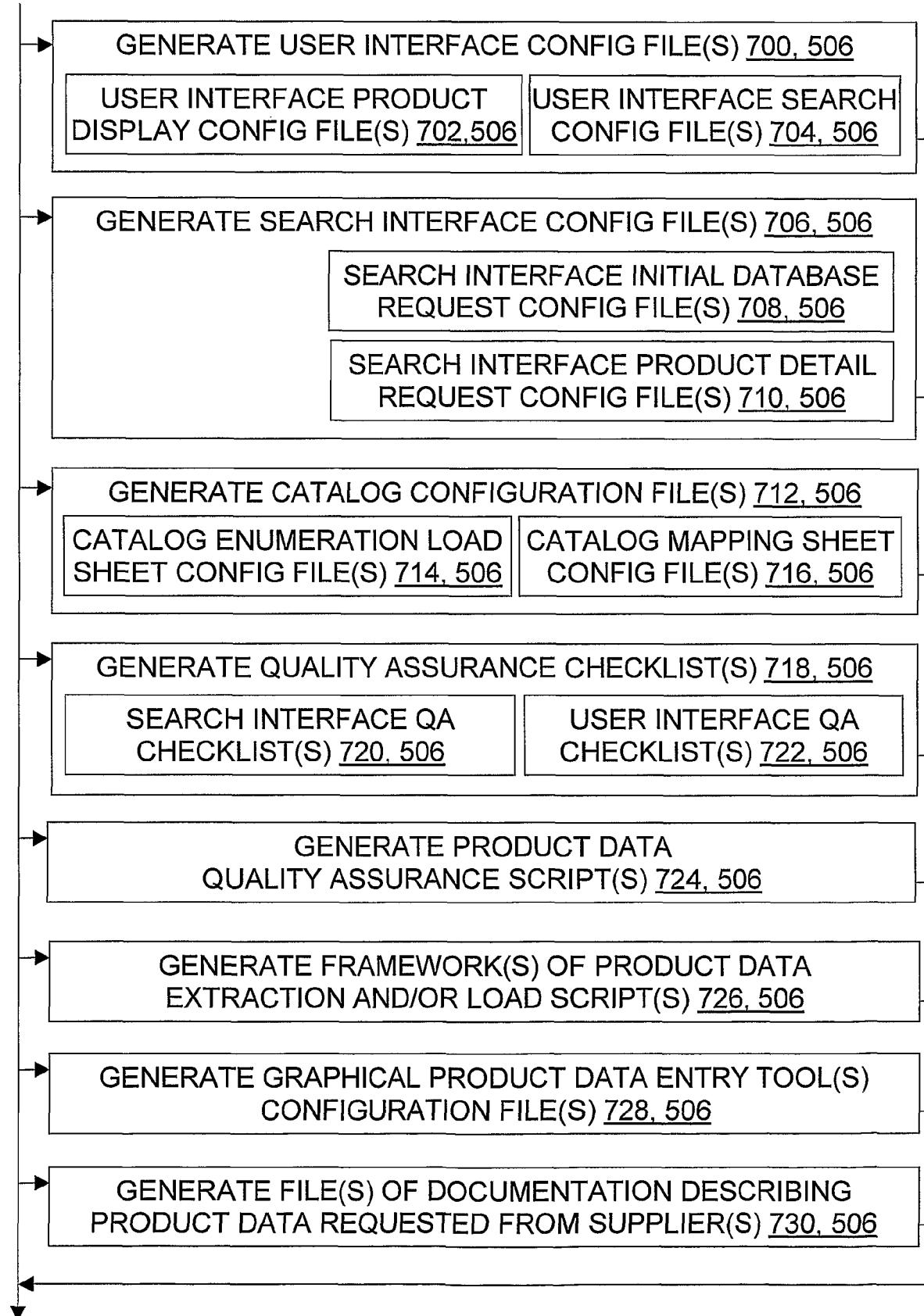


Fig. 7

6/8

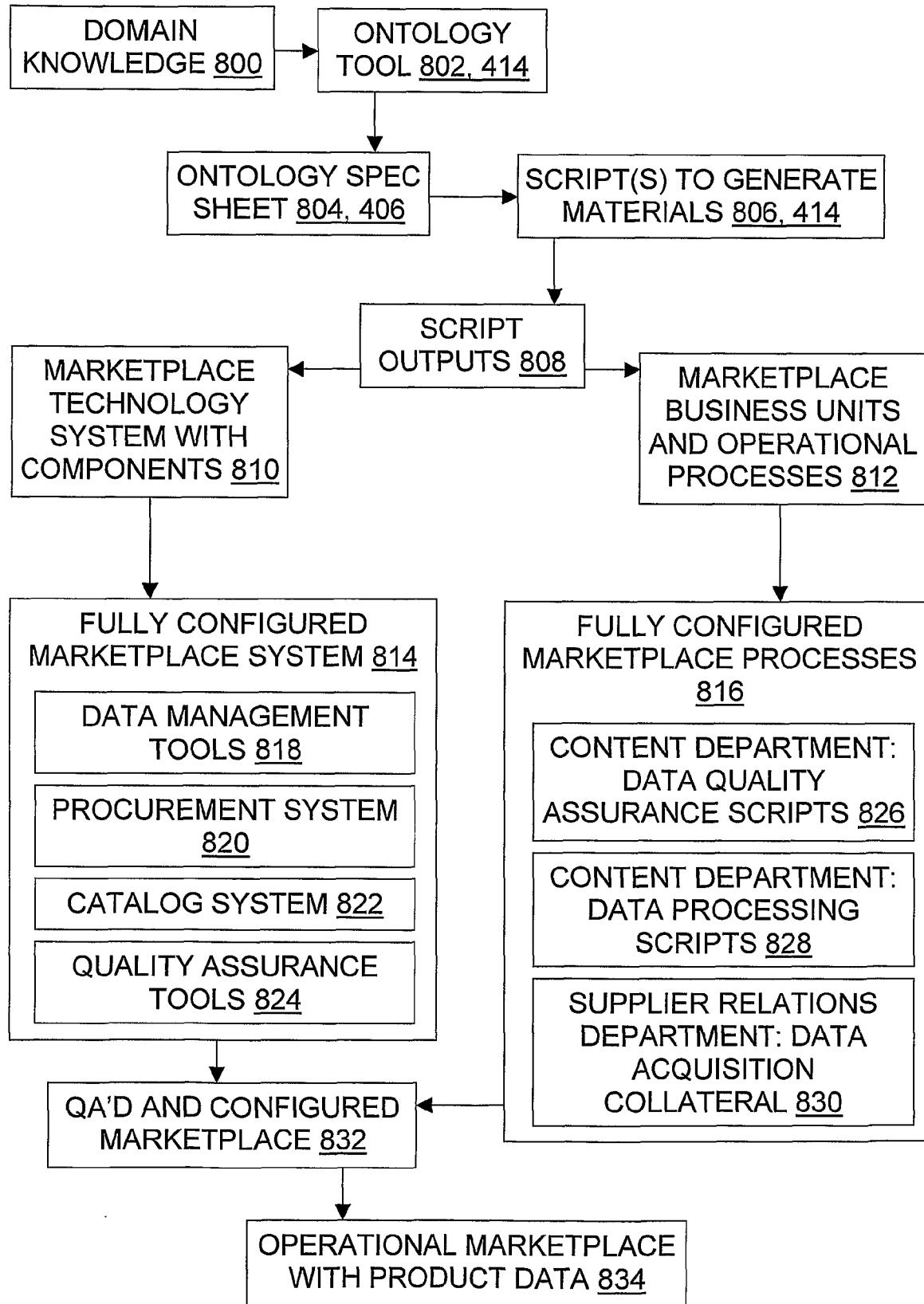


Fig. 8

7/8

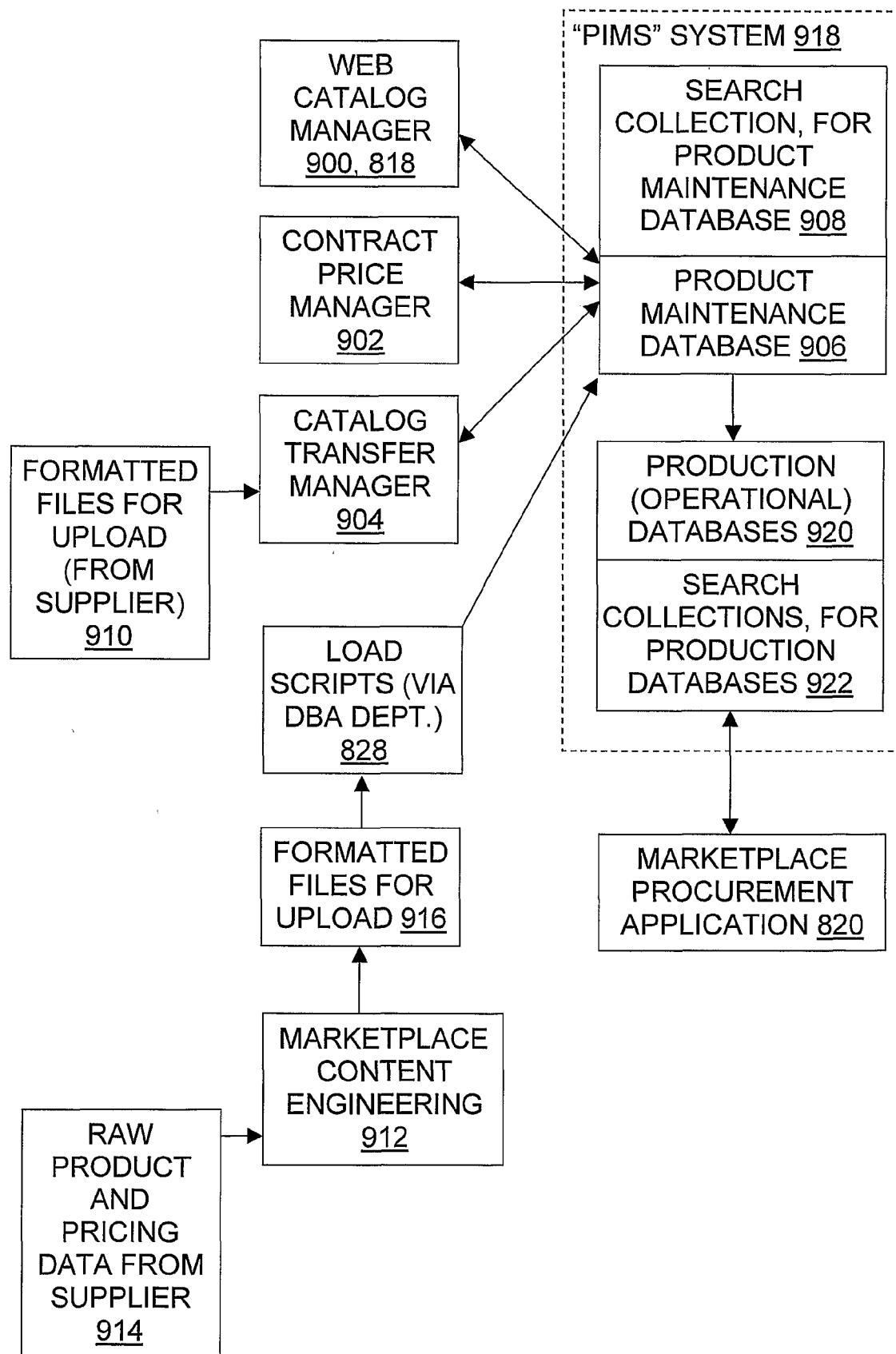


Fig. 9

8/8

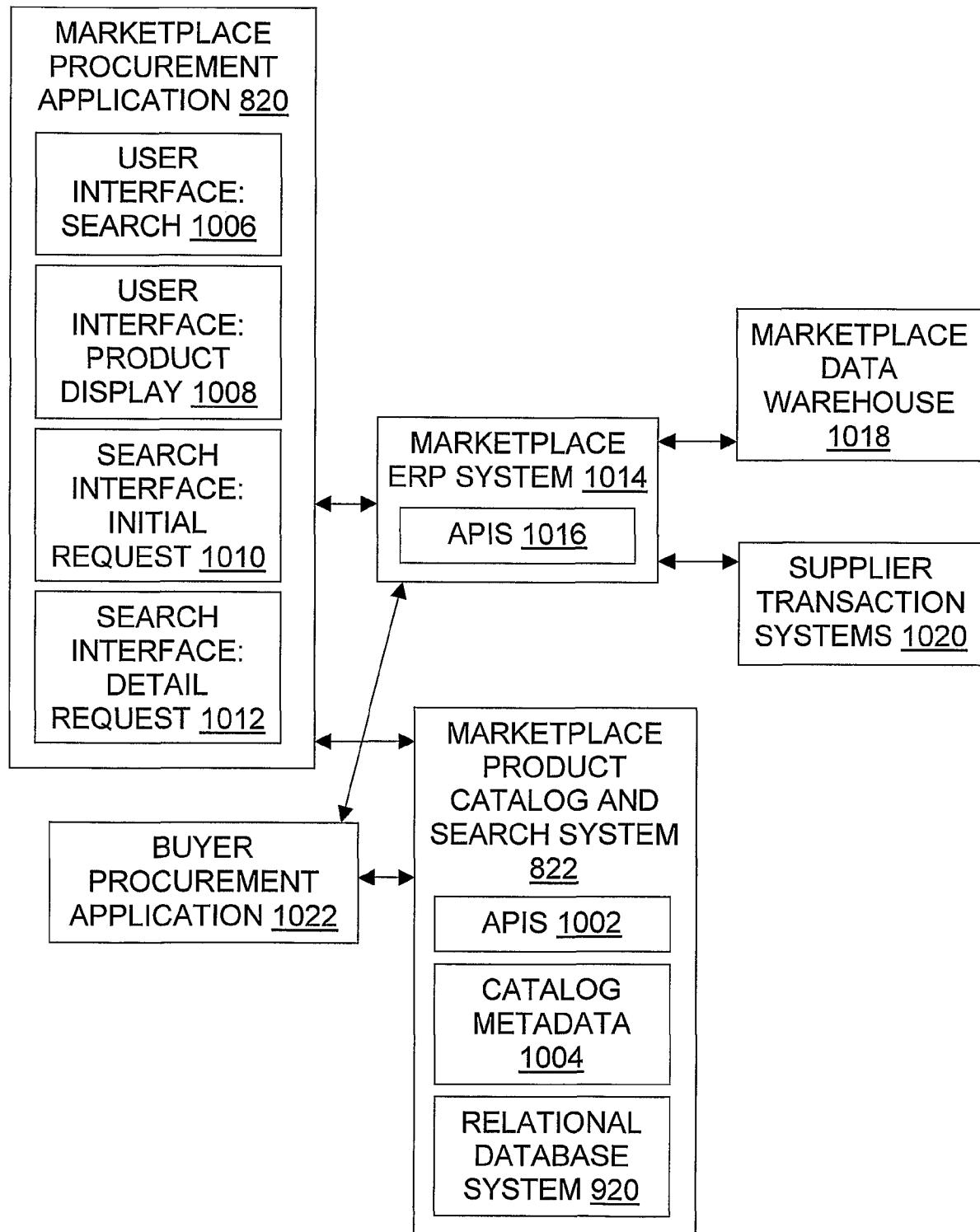


Fig. 10

## INTERNATIONAL SEARCH REPORT

International application No.  
PCT/US01/25468

## A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 17/60

US CL :705/26

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 705/26, 27

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,740,425A (PROVILUS) 14 April 1998; see at least abstract; Fig. 14 (all); col. 3, lines 9-59; col. 5, lines 66-67; col. 6, lines 1-11; col. 50, lines 11-67; col. 51, lines 1-67; col. 52, lines 1-67; col. 53, lines 1-67; col. 54, lines 1-65	1-63
Y	US 6061675A (WICAL) 9 May 2000; see at least abstract; Fig. 2; Fig. 4; Fig. 7; Fig. 8 (1000); col. 1, lines 29-31; col. 1, 66-67; col. 2, lines 1-39)	1-63
X, E	"Zycus Commits to Join PeopleSoft's Network of Content Service Providers, Objective Focuses on Solutions That Provide Catalog Content Integration," Business Wire, 27 August 2001; see all pages.	1-63

Further documents are listed in the continuation of Box C.  See patent family annex.

* Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A" document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E" earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O" document referring to an oral disclosure, use, exhibition or other means		
"P" document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search

17 OCTOBER 2001

Date of mailing of the international search report

19 NOV 2001

Name and mailing address of the ISA/US  
Commissioner of Patents and Trademarks  
Box PCT  
Washington, D.C. 20231

Facsimile No. (703) 305-8230

Authorized officer  
ROBERT MADISON POND  
*Peggy Harold*

Telephone No. (703) 605-4253

## INTERNATIONAL SEARCH REPORT

In \_\_\_\_\_ international application No.

PCT/US01/25468

## C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	"Simulation of Business Processes," Paul, Ray J., American Behavioral Scientist, August 1999, v42, n10, p1551-1576.	1-63
Y	"Share the Ontology in XML-based Trading Architectures," Smith, Howard, Communication of the ACM, March 1999, v42, n3, pp110-111; see pp 110-111.	1-63

# INTERNATIONAL SEARCH REPORT

International application No.  
PCT/US01/25468

## B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

East (US Patents, JPO, EPO, Derwent, IBM, US Patent Pubs)

Dialog (Dialindex: alleng, allproducts)

search terms: ontology, catalog content integration, web developmnet, tools